

# Предисловие

---

От того, как команда организует работу, во многом зависит то, чего она достигнет. Процессы, методы, практики, подходы, дисциплина, стиль общения, настрой. Ответ на вопрос «как?» — ключ к успеху любого дела.

Часто говорят: «Результат важнее всего». Но это не всегда так. Точнее, почти всегда не так.

Если говорить о разработке программных продуктов, достижение результата не тем путём часто приводит команду к выгоранию, недоверию, и в итоге к развалу. Да, работа сделана, но какой ценой? Как мы выдержали всё это? Неужели нужно проходить через те же мучения снова и снова, год за годом?

Посчитайте, сколько у вас было проектов, за которые вы с радостью взялись бы снова? И наоборот, на скольких проектах сроки были сорваны, люди выгорели, ожидания руководства не имели ничего общего с реальностью? Сколько раз люди ссорились, разочаровывались и уходили?

Да, иногда результат — «всё». Совсем «всё».

Тогда какой путь — правильный?

Мы в Basecamp часто замечаем интерес к тому, как организована наша работа. Люди удивляются — как у нас получается делать так много, так быстро, такой небольшой командой. И почему никто не выгорает и не увольняется.

Во-первых, мы не адепты Waterfall, Agile или Scrum. Во-вторых, мы не обклеиваем стены заметками. В-третьих, мы не проводим ежедневные стендапы, спринты, и вообще ничего, что хотя бы отдалённо предполагает работу на износ. Никаких бэклогов, Канбан-досок, расчётов личной эффективности, ничего такого.

Мы описывали некоторые из наших методов в блогах, мастер-классах и конференциях, но не сводили их в единую систему. В этой книге свели.

Мы подробно описали наши рабочие процессы и готовы поделиться ими со всеми, кому они интересны. С теми, кто неравнодушен к тому, как ещё может быть, кто хочет работать лучше, чем сейчас.

Эта книга — маяк. Ваша команда достаточно блуждала в темноте. Мы предлагаем новую дорогу и надеемся, путешествие вам не только понравится, но и принесёт реальную пользу.

Приятного чтения.

## Слова благодарности

---

Джейсон (Jason Fried) и Дэвид (David Heinemeier Hansson), основатели компании Basecamp, заложили основу для этой книги. Она продиктована их ценностями, культурой Basecamp, и конечно, пятнадцатью годами проб и ошибок.

Неоценимый вклад внесли Bob Moesta и Chris Spiek — без них книга бы не состоялась.

Лекции Yaneer Bar-Yam в New England Complex Systems Institute помогли мне найти нужную структуру для описания метода Shape Up.

Разные команды в Basecamp пробовали, проверяли и улучшали процессы, описанные в книге, в течение многих лет, на реальных продуктах. Книга опирается на реальность, и это — их заслуга.

# Вступление

---

Эта книга — о том, как мы в Basecamp разрабатываем продукты. В ней описаны инструменты, которые вы можете использовать в своих рабочих процессах.

Вероятно, вы читаете эту книгу, поскольку столкнулись с типичными проблемами компаний, разрабатывающих программные продукты.

## Болезни роста

По мере того, как команды растут, растут и проблемы:

- проекты, кажется, никогда не заканчиваются;
- продакт-менеджеры не могут найти время спокойно подумать о стратегии и будущем продукта;
- владельцы компании обнаруживают, что новые фичи выпускаются все медленнее, не то что в прежние времена.

Мы в Basecamp не избежали этих проблем — в какой-то момент из 4 человек мы выросли до 50.

Продукт Basecamp был создан нами в 2003 году для самих себя. В то время мы делали веб-сайты на заказ. Клиент, дизайнер и менеджер постоянно играли в испорченный телефон, теряя важную информацию. Мы хотели, чтобы Basecamp стал единым местом для обсуждения сделанной и будущей работы.

Оказалось, что у многих компаний похожие проблемы — информация «уходит в песок». Сегодня миллионы людей всех

возможных профессий используют Basecamp как основной источник правды.

Первую версию Basecamp создала команда из трёх человек. Джейсон (Jason Fried), основатель, отвечал за дизайн. Дэвид (David Heinemeier Hansson) — за разработку (и как побочный продукт создал Ruby on Rails). В то время я работал дизайнером сайтов и интерфейсов. Я брал направление, обозначенное Джейсоном, и вместе с ним прорабатывал все детали.

С июля 2003 до запуска в феврале 2004 Дэвид работал только 10 часов в неделю. Мы знали, что справимся, только если эти 10 часов будут тратиться очень обдуманно. Приём «вколачивания» объёма работы во временные рамки был придуман как раз по этим причинам.

Я изучал техники, которые программисты используют для управления сложностью — факторинг, уровни абстракции, разделение ответственостей (SoC) — и решил попробовать применить их к дизайну и управлению продуктом.

Первая проверка идеи состоялась в 2009 году. К тому времени в компании было несколько программистов, мы продавали 4 отдельных продукта. Мы захотели объединить продукты в один, с единой авторизацией и оплатой. Это был огромный технический сложный проект с заковыристыми сценариями. Нам предстояло не только аккуратно переделать архитектуру, но и заставить всех поменять логины и пароли — необходимость этого непросто объяснить пользователям. Я был дизайнером и продукт-менеджером, поэтому опробовал техники «макетной платы» (breadboarding) и «карты проекта» (scope mapping), описанные в этой книге.

Результаты были настолько хороши, что мы решили использовать те же техники в 2012 году для полной переделки Basecamp 2.0. Вновь проект был технически сложен, и вновь работа прошла на удивление гладко.

К 2015 году в компании было много новых сотрудников, не имевших подобного опыта. Оказалось, что передать его не так просто. Продуктовая команда выросла в 4 раза и все работали удалённо. Перенимать знания «лицом к лицу» не получалось. Нам нужно было научиться формулировать, что и как именно мы делаем.

В то время я отвечал за продуктовую стратегию. Мне нужны были новые термины, чтобы описать наши процессы. Мы описали «формирование проекта» (shaping), «презентацию» (pitching) и «голосование» (betting).

По мере того, как мы документировали эти процессы, люди вокруг стали интересоваться, как именно это всё работает. Однажды Джейсон отозвал меня в сторонку и сказал: «Тебе стоит написать об этом книгу».

Вот она. По сути, тут две книги в одной. Первая — набор понятий для работы с рисками, неопределённостями и трудностями, возникающими у всех, кто разрабатывает продукт. Вторая — описание конкретных практик, которые мы используем для развития наших продуктов с учётом текущего размера компании.

Главные идеи книги:

## 1. Циклы в 6 недель

Мы работаем в рамках **6-недельных циклов**. Это достаточно много, чтобы разработать что-то осмысленное, и достаточно мало, чтобы все видели финиш уже на старте. Так гораздо легче использовать время обдуманно. Почти все наши проекты выпускаются в течение 6-недельного цикла.

Основа наших решений — желание улучшить продукт за 6 недель. Никакого управления временем, подсчёта рабочих часов, отчётов за день. У нас нет ежедневных летучек.

Мы не пересматриваем план работ каждые 2 недели. Вместо этого мы говорим: «Если этот проект выпущен за 6 недель, время потрачено не зря». Затем мы оставляем команду в покое, чтобы она спокойно достигла результата.

## 2. Формирование проекта

Мы **формируем проект**, прежде чем отдать его команде. Небольшая команда менеджеров прорабатывает все основные идеи будущих проектов, в то время как программисты работают над проектами, сформированными ранее. Важно, чтобы уровень проработки идей был золотой серединой — не слишком абстрактный (чтобы команды понимали, что нужно делать), но и не слишком конкретный (чтобы у команды была достаточная свобода в принятии решений).

Когда мы формируем проект, вместо оценки времени мы используем понятие «аппетит». Вместо «сколько времени это займёт?» мы спрашиваем «сколько времени мы хотим на это потратить?».

В этом основная идея формирования как процесса — сформулировать проблему и предложить решение так, чтобы

проект вместился в наш «аппетит».

### **3. Ответственность команды**

Мы перекладываем всю ответственность за проект на небольшую команду дизайнеров и программистов. Они сами ставят себе задачи, меняют объем работы, разбивают проект на части и работают над ними по своему усмотрению. Этот подход радикально отличается от общепринятого подхода, при котором менеджеры нарезают проект на задачи (тикеты) и раздают их разработчикам.

Вместе эти практики усиливают друг друга. Команды работают более автономно — у менеджеров и аналитиков уходит меньше времени на управление командами, появляется время лучше сформировать будущие проекты. К лучше сформированным проектам у команды меньше вопросов — она работает более автономно.

### **4. Управление рисками**

На каждом этапе цикла мы управляем одним конкретным риском — не выпустить проект вовремя. В этой книге нет ничего про риск разработать ненужный проект (про это написано много других книг, рекомендуем [Competing Against Luck](#)). Возможно, у вас лучшие идеи на свете, но если вы не можете их воплотить, в чем их смысл?

Эта книга — о риске зайти в тупик, не успеть к сроку, потратить время на внезапные проблемы, и в итоге иметь планы, но не иметь времени на их воплощение.

На этапе *формирования* мы уменьшаем риск тем, что решаем все открытые вопросы верхнего уровня **до** передачи команде в работу. Проект не будет отдан, пока в нем есть видимые дыры или нечёткие зависимости.

На этапе *планирования* мы уменьшаем риск тем, что ограничиваем себя шестью неделями. Если проект не готов к сроку, обычно он отменяется. Этот «предохранитель» даёт нам уверенность, что мы не потратим в разы больше, чем планировали, на что-то, что часто требует пересмотра идеи.

На этапе *разработки* мы уменьшаем риск тем, что дизайнер и программист работают вместе с самого начала. Вместо отдельных заданий, которые, если повезёт, соединяются в последний день в работающий проект, команда разбивает проект на осмысленные части и выпускает их по одной, не дожидаясь конца цикла. Обычно команда начинает с самых непонятных частей и, выпуская часть за частью, получает возможность мгновенно понять, что в целом может пойти не так.

## Как устроена эта книга

Часть 1 посвящена **формированию** — подготовительной работе, после которой проект готов к планированию. Каждая глава описывает этап процесса — определение «аппетита», описание решения, презентация. В этой части вы узнаете про конкретные приёмы (например, макетные платы и наброски толстым маркером), позволяющие держать нужный баланс абстракции.

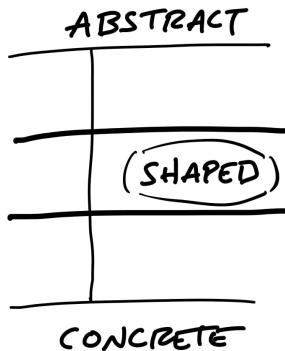
Часть 2 – про **голосование**. Как мы выбираем, что взять в работу.

Часть 3 – про **разработку**. Чего мы ожидаем от команд, какими процессами они пользуются, чтобы понять, что делать, что уже ясно, а что ещё неясно, как взаимодействуют дизайн и разработка, и чем жертвовать, чтобы закончить работу в срок.

Наконец, Приложение – для тех, кто хочет начать менять работу в своей компании. Советы тем, кто хочет попробовать 6-недельные циклы, адаптировать наши техники к своему размеру компании, а также использовать для их применения наш продукт, Basecamp.

# Формирование проекта

---



Когда мы формируем проект, мы ищем золотую середину — не слишком абстрактно, не слишком конкретно.

## Схемы экранов — слишком конкретно

Начиная работу сразу со *схем экранов* (wireframes), вы слишком рано принимаете слишком много решений. Дизайнеру не остается пространства для манёвров. Один знакомый сформулировал так:

Я даю схему экрана дизайнеру и говорю ей: «Я, конечно, нарисовал вот так, но это не то, что я хочу от тебя получить. Придумай, как это сделать по-другому!» А это непросто, если за тебя уже всё нарисовали.

Слишком конкретные требования к дизайну также ведут к неверным оценкам объёма работы. Звучит странно, но чем конкретнее задание, тем сложнее его оценить — подгоняя

решения к требованиям, команда сталкивается со сложностями, которые были не видны в начале. Когда задание звучит как «сделай вот так», пропадает возможность пересмотреть решения, не стоящие усилий.

## Слова — слишком абстрактно

Другая крайность — проекты, которые сформулированы парой строк текста. Никто не сможет осознать, в чем именно состоит проект. «Создать календарь» или «добавить уведомления в группы» — вроде понятно, но что конкретно? Команда не понимает, что точно должно входить в проект, а чего, наоборот, делать не надо. Другой знакомый сказал об этом так:

Приходится читать мысли, делать что-то, потому что «мы поймём, чего хотим, когда увидим это».

Оценки таких проектов также выходят из-под контроля. Нет границ, позволяющих понять, где нужно остановиться.

## Пример из жизни: календарь с точками

Вот пример удачно сформированного проекта.

Мы запустили Basecamp 3 без календаря. Продукт показывал события простым списком, без какой-либо сетки по дням, неделям или месяцам.

Клиенты сразу же стали просить добавить календарь. Мы разрабатывали календари раньше и знали, насколько это сложно. Можно спокойно потратить 6 месяцев на нормальный календарь, в котором были бы:

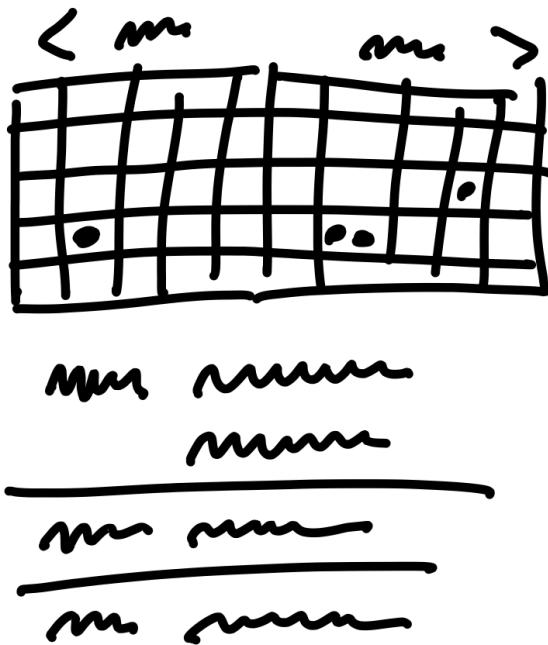
- перетаскивание событий по сетке;
- перенос длинных событий на новую строку;
- разные отображения для месяцев, недель, дней;
- изменение длительности события путём перетаскивания его за угол;
- разные цвета для разных категорий событий;
- разные поведения для компьютеров и мобильных устройств.

В прошлой версии Basecamp был календарь, но только 10% пользователей им пользовалось. Вот почему у нас не было *аппетита* потратить 6 месяцев на его разработку. С другой стороны, мы были бы готовы потратить 6-недельный цикл одной команды, если бы нашли другое решение проблемы. Это примерно одна десятая часть работы, которую представляют люди, когда говорят «календарь». Весь вопрос в том, какая именно десятая.

Мы провели исследование (о нём в следующей главе) и более чётко сформулировали проблему, которую хотели и могли решить. Нас вдохновил интерфейс календарей на телефонах с сеткой на 2 месяца, без интерактива. Каждое событие отмечается точкой под числом. Название события — в списке под сеткой, и нажатие на точку перематывает список до соответствующего события. Мы назвали проект «Календарь с точками».

Это не был полноценный календарь. Никакого перетаскивания событий. Вместо переноса длинных событий просто повторяем точки. Никаких категорий и разных цветов. Мы легко пошли на все эти жертвы, поскольку чётко представляли себе, какую проблему решаем.

Вот картинка, которая формулирует проект:



Обратите внимание, насколько груб набросок, сколько деталей упущено. У дизайнера была свобода в принятии конкретных решений по интерфейсу.

Но в то же время основная идея описана очень чётко. Ясно, как всё должно работать, что делать, и чего не делать.

Конечный результат работы команды выглядел так:

August							September						
SUN	MON	TUE	WED	THU	FRI	SAT	SUN	MON	TUE	WED	THU	FRI	SAT
			1	2	3	4	5					1	2
6	7	8	9	10	11	12	3	4	5	6	7	8	9
13	14	15	16	17	18	19	10	11	12	13	14	15	16
20	21	22	23	24	25	26	17	18	19	20	21	22	23
27	28	29	30	31			24	25	26	27	28	29	30

**Wed, Aug 23** Interview with The TaTa Top 🎙  
9:30am - 10:00am

Interview with Tim Krepp 🎙  
12:00pm - 12:30pm

[Add an event](#)

**Mon, Aug 28** Interview with Amy Trager 🎙  
11:00am - 11:30am

**Wed, Aug 30** Interview with Jenifer of Colorstock 🎙  
10:00am - 10:30am

**Fri, Sep 1** Dr. Bronner's visit 🎙

Важные свойства сформированного проекта:

## 1. Незаконченность

По виду сформированного проекта ясно, что это черновик. Всем видны белые пятна, которые они заполнят своей работой. Слишком подробно сформированный проект отвлекает всех на ненужные в данный момент детали.

## 2. Продуманность

Несмотря на очевидную незаконченность, сформированный проект тщательно продуман. Работа не разбита на части,

но общее решение описано чётко и полно. Всем ясно, куда двигаться, даже если в процессе работы возникнут сюрпризы. Все возникшие вопросы решены.

### **3. Границы**

Сформированный проект также описывает, чего *не надо* делать, где нужно остановиться. У команды есть чёткое ограничение — время, которое она может потратить. Чтобы уложиться в это время, нужно знать, что оставить за бортом.

Итак, границы проекта дают команде свободу принятия решений, а продуманность решения и ограниченность служат регуляторами, которые не позволят команде сделать ненужное или зайти в тупик.

## **Кто формирует проект**

Формирование проекта — креативная работа, но в то же время она требует учёта технических ограничений и приоритетов бизнеса. Вам нужно либо погрузиться во все эти области, либо работать в команде с ещё одним-двумя людьми.

На практике формирование проекта — в основном дизайн. Результат этой работы — понимание того, как с продуктом (или его частью) взаимодействует пользователь.

Чтобы формировать проект, не нужно быть разработчиком, но нужно понимать разработку — знать, что возможно, что трудно, а что легко. Это необходимо, чтобы заранее видеть возможности и препятствия в реализации проекта.

Это также аналитическая работа, требующая критического подхода. В чём именно проблема, которую решает проект? Почему она важна? Что именно считать успехом? Кого коснётся проект? Чем придётся пожертвовать, если взять проект в работу?

Как часто бывает в начале творческого процесса, формирование проекта — закрытый процесс. Вы набрасываете идеи в одиночку или с одним-двумя коллегами, на бумаге или доске. Пока ещё никто другой не сможет понять, что вы имеете в виду, просто посмотрев на эти наброски. Вы движетесь быстро, без сожаления отметая одну идею за другой.

## **Два потока**

Нельзя чётко запланировать формирование проекта, поскольку по своей природе эта работа полна неопределённостей. Поэтому у нас есть два отдельных потока — один для формирования, другой для разработки. В течение каждого цикла команды разработки работают над ранее сформированными проектами, а команды менеджеров формируют проекты для будущих циклов. Эта работа не обсуждается с другими командами, пока не будет принято решение передать сформированный проект в работу. Поэтому, если проект не складывается, у менеджера есть варианты — положить проект на полку или отменить.

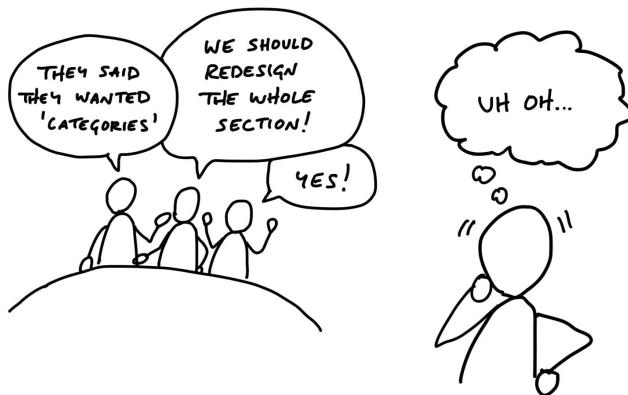
## **Этапы формирования проекта**

Вот 4 основных этапа, которые мы разберём в следующих главах.

- 1. Обозначьте границы.** Мы формулируем проблему и оцениваем, насколько важно её решить.
- 2. Набросайте элементы решения.** Мы используем более абстрактные методы, чем схемы экранов, и это даёт нам возможность быстрее рассмотреть множество идей.  
Результат — краткое, без подробностей, описание проекта, который решает проблему и может быть реализован в заданные сроки.
- 3. Оцените риски.** Когда у нас есть решение, мы смотрим, что может пойти не так — ищем скрытые трудности или зависимости, которые застопорят работу. Далее либо урезаем решение, либо предлагаем способы уменьшить риски.
- 4. Напишите презентацию.** Когда проект кажется готовым к передаче в работу, мы пишем формальный документ — презентацию. Она включает в себя описание проблемы, ограничений, предложенное решение, риски, ответы на найденные заранее вопросы. Презентации уходят на голосование — и если проект одобрят, презентация станет основой для работы команды разработки.

# Обозначьте границы

---



Первый этап формирования — обозначить границы того, что мы собираемся делать. Разговоры о мелком изменении радикально отличаются от разговоров о глобальной переделке.

Обычно все начинается с сырой идеи, например, «пользователи просят добавить уведомления в группы». Прежде чем бросаться обсуждать все возможные варианты реализации, стоит обозначить границы обсуждения, чтобы сделать его продуктивнее.

## Определите *аппетит*

Даже самую соблазнительную на первый взгляд идею стоит проверить — стоит ли инвестировать в неё время. Если безусловно считать идею ценной, мы либо слишком поспешно

выделим и потратим ресурсы, либо будем долго обсуждать множество вариантов реализации, и на этом всё закончится.

Иногда идеи не вызывают энтузиазма. Клиент хотел календарь, мы не особо хотели его разрабатывать, но чувствовали, что с запросом нужно что-то делать.

В обоих случаях полезно чётко определить, сколько времени и внимания мы готовы потратить на конкретную идею.

Достаточно ли быстрой правки? Стоит ли идея полного цикла?  
Много ли придётся переделывать в продукте?

Мы называем это *аппетит*. Можно сказать, что это бюджет времени для команды стандартного размера. У нас обычно два значения:

- *маленький аппетит* — 1-2 недели для команды из 1 дизайнера и 1-2 программистов. Несколько таких проектов складываются в 6-недельный цикл;
- *большой аппетит* — 6 недель для той же команды.

Изредка, когда объём работы заметно больше цикла, мы стараемся влезть во временные рамки, сузив определение проблемы. Если и это не помогает, разбиваем проект на отдельные проекты, которые совпадают с 6-недельным циклом.

## **Фиксированное время, гибкий объем работы**

Аппетит противоположен привычной «оценке времени». Оценка времени — это «сколько времени уйдёт на данную

задачу». Аппетит — это «как сформулировать задачу, на которую уйдёт данное время». Аппетит — ограничитель в творческом процессе.

Этот принцип — «фиксированное время, гибкий объем работы» — ключ к успеху формирования и завершения проектов. Например, эту книгу было бы сложно написать, если бы я знал, что всегда могу добавить что-то, объяснить что-то другими словами. Когда есть чёткий дедлайн, внезапно приходится принимать решения. Если у меня осталась неделя, я могу добавить главу, или вычитать текст на опечатки. Это — компромисс между временем, качеством и объемом. Я не готов пожертвовать качеством, поэтому уменьшаю объем, не добавляя главу. Без ограничения по времени, мне не пришлось бы искать компромисс. Если бы объем был фиксирован, мне *пришлось бы* добавить главу, пожертвовав качеством.

Мы применяем этот принцип на каждом этапе работы, от формирования проектов до разработки и выпуска. Сначала аппетит ограничивает нас в том, что именно мы хотим сделать. Затем, когда проект передан в разработку, аппетит заставляет команду решать, что важно, а чем можно пожертвовать.

## **Хорошее относительно**

Не бывает «наилучших» решений. На выбор решения всегда влияют ограничения. Если время не ограничено, всегда можно найти решение лучше. Обед из 10 блюд в теории — наилучшее решение проблемы голода, но если у вас нет ни денег, ни времени, берите хот-дог.

Разные размеры аппетита приводят к совершенно разным решениям в течение проекта. Можем отобразить данные из таблицы адаптивно, с отдельным дизайном для данных из каждой колонки. А можем ограничиться единым стилем для всей таблицы. Решать, насколько идея «хороша», мы можем только зная, насколько она важна и сколько мы готовы выделить времени.

## **Обсуждение сырых идей**

Наша стандартная реакция на любую сырую идею — «интересно. Может быть, какнибудь». Мы оставляем себе выбор. Мы не добавляем идею в список задач (бэклог). Мы даём идее время доказать, что она важна.

Рано говорить «да» или «нет» в первом обсуждении. Даже если идея кажется отличной, не стоит брать на себя обязательства, которые мы до конца не осознали. Для начала идею нужно **сформировать**. Если говорить «да» сразу, результатом будет вышедший из-под контроля список задач.

Важно не давать волю эмоциям. Не стоит отказываться от идеи сразу только потому, что она не до конца понятна. В дальнейших обсуждениях появиться информация, которая заставит взглянуть на неё по-новому.

С другой стороны, проявить энтузиазм в первом разговоре означает слишком рано дать сигнал — отлично, надо брать. Вполне возможно, что позже, собрав все идеи на голосование, вы поймёте, что есть более важные проекты.

## **Уточнение проблемы**

Вместе с определением аппетита, мы стараемся уточнить проблему — возможно, реальная проблема в другом.

Однажды клиент попросил нас добавить более сложные права доступа к файлам. Этот проект легко съел бы полный 6-недельный цикл. Вместо того, чтобы сразу взяться за проблему, мы решили узнать причину запроса. Оказалось, что кто-то «архивировал» файлы, не осознавая, что они пропадут у всех в компании. Вместо того, чтобы создавать сложную систему прав доступа, мы добавили предупреждение, которое объясняет поведение архивирования. Это работа на один день, не на 6 недель.

Ещё один пример — «Календарь с точками» из предыдущей главы. Каждый примерно понимал, что такое календарь, но обсуждение идеи выявило массу неопределённости, которая кардинально влияла на оценку объёма. Если мы хотим потратить только 6 недель, а не 6 месяцев, как быть?

Итак, вместо вопроса «Что нужно сделать?» мы начинаем с вопроса «Что конкретно не так?». Конечно, круто иметь полноценный календарь. Но зачем конкретно он нужен тем, кто его просит? В какой точке их работа встаёт из-за его отсутствия?

## **Пример из жизни: определение «календаря»**

В случае календаря, мы созвонились с клиентом и спросили, *в какой момент* ей нужен календарь (но не спрашивали, почему он ей нужен и как он должен выглядеть).

Оказалось, что она работает в офисе с большим настенным календарём, на котором коллеги отмечают время, когда переговорки заняты. Однажды она работала из дома, и заказчик попросил её назначить встречу. Ей пришлось ехать в офис, чтобы посмотреть на стену, поскольку она не могла узнать свободное время через компьютер из дома.

На самом деле, её потребность была не «перенести календарь в компьютер», не «делать всё, что может делать календарь», а «иметь возможность видеть свободные места в календаре».

Эта история дала нам **основу** для дальнейшей работы. В Basecamp был список событий, но его было трудно использовать для планирования ресурсов, поскольку свободных мест не было видно. У нас ещё не было решения, но мы смогли уточнить проблему так, чтобы появился интерес решить её в рамках аппетита. Решение пришло в виде «Календаря с точками», описанном в прошлой главе.

А если не получается выявить проблему? Аппетит также ограничивает нас в исследованиях. Если формулировка проблемы не складывается, мы откладываем идею и работаем над чем-то ещё. Возможно, в будущем новая информация позволит взглянуть на проблему под другим углом.

## **Берегитесь кота в мешке**

Самые худшие виды размытых идей — это «редизайны» или «рефакторинги», которые не продиктованы никакой проблемой. Идея вида «нам нужен редизайн раздела Файлы» — это не проект, а кот в мешке. Будет очень трудно определить, что это означает, где оно начинается и где заканчивается.

Вот более продуктивный подход: «отправка сразу нескольких файлов сейчас занимает слишком много шагов». Про эту формулировку уже можно задавать вопросы: что именно не работает? в каком контексте получается слишком много шагов? какие части текущего дизайна придётся переделывать, а какие нет?

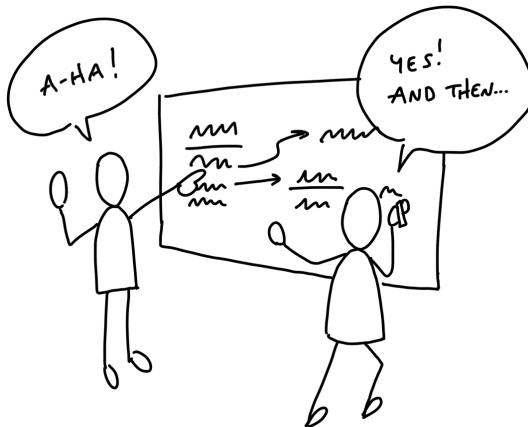
Кота в мешке часто можно узнать по «2.0» в названии. Однажды мы совершили ошибку, запустив проект «Файлы 2.0», не разобравшись толком, что мы имели в виду. Так соблазнительно звучала идея переделать огромный кусок приложения! Мы знали, что в этой части много проблем, но не разобрались, что именно мы хотим сделать. Проект провалился, поскольку мы не знали, как должна выглядеть завершённая работа. Мы исправили ситуацию, разделив проект на несколько мелких — «Улучшенные превью», «Разные цвета папок» — и чётко сформировав каждый проект.

## **Границы на замке**

Когда у нас есть все три вещи — сырья идея, аппетит, уточнённая проблема — мы готовы двигаться дальше и искать решение.

## Найдите элементы решения

---



Когда мы ограничили наш аппетит и определили проблему, которую решаем, время перейти от слов к делу (то есть от сырой идеи к интерфейсному решению). Искать решение проблемы можно десятками разных способов. Нам важно двигаться быстро и пробовать совсем разные идеи, не погружаясь сразу глубоко.

## Выберите правильную скорость

В этом нам помогают две вещи.

Во-первых, собираем только действительно нужных людей — или никого. Менеджер, формирующий проект, либо работает один, либо с тем, кто может полноценно участвовать в поиске решения. У этого человека уже есть нужные знания, его

не требуется вводить в курс дела, он честен и может быстро отбраковывать идеи.

Во-вторых, избегаем подробностей в набросках. Если сразу рисовать полноценные схемы экрана, мы не заметим, как погрязли в деталях и упустили время обработать широкий спектр разных идей.

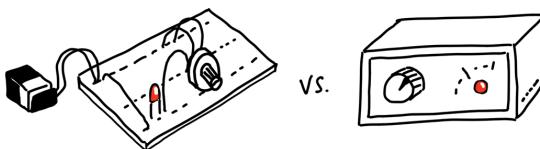
Фокус в том, чтобы проработать каждую идею **минимально** глубоко. Ищем ответы на вопросы:

- куда в текущий продукт будет добавлена наша новая идея?
- какой путь будет к ней вести?
- каковы её основные части или сценарии?
- куда она в конце приводит?

Для этой работы у нас есть два отличных приёма: **макетные платы** и **толстые маркеры**. С их помощью мы делаем наброски сценариев, обсуждаем плюсы и минусы, будучи уверенными, что говорим об одном и том же.

## Макетные платы

Идея была подсмотрена в схемотехнике, и означает прототип устройства, в котором все компоненты соединены друг с другом, но не в готовом оформлении, а на временной плате.



Разговор на уровне «а не подключить ли нам индикатор и регулятор» сильно отличается от обсуждения материала коробки, расположения регулятора слева или справа от лампочки, скругления углов устройства, и тому подобного.

Мы набрасываем ключевые компоненты и связи в интерфейсе, который придумываем, не затрагивая его внешний вид. Рисуем только:

1. **«Места».** Это части интерфейса, в которые можно попасть — экраны, диалоговые окна, меню.
2. **«Действия».** Это части интерфейса, с которыми можно взаимодействовать — кнопки, поля форм, и т.д.  
Мы считаем экранный текст «действием» — читая его, пользователь получает информацию для будущих действий.
3. **«Связи».** — Это линии, которые показывают, как действия переносят пользователя с одного места в другое.

В набросках не используем картинки, только слова! Значение имеют только компоненты (места, действия) и их связи.  
Набросав их, мы быстро поймём, решает ли этот набор действий нашу проблему.

## Пример

Предположим, у нас продукт для создания счёта-фактур (инвойсов). Мы хотим добавить Автоплатёж, чтобы клиенты наших клиентов могли оплачивать будущие инвойсы автоматически.

Как включить Автоплатёж? Что для этого требуется? Начнём с того, где пользователь встречается с Автоплатежом. Пусть это будет страница Инвойса. Пишем и подчёркиваем.

INVOICE

Дальше, пусть на этой странице появится новая кнопка «Включить Автоплатёж». Это действие. Пишем его под названием места.

INVOICE

TURN ON  
AUTORACH

Куда ведёт эта кнопка? В какое-то место для настройки Автоплатежа. Мы не решаем, будет ли это новая страница, модальное окно или ещё что-нибудь. С точки зрения нашей «топологии» это неважно. Рисуем связь действия с новым местом.

INVOICE

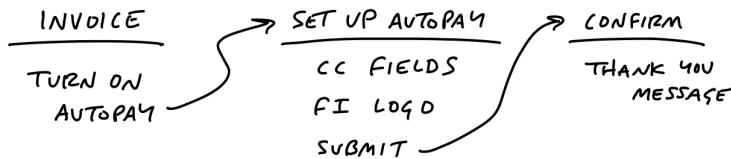
TURN ON  
AUTORACH

SET UP AUTORACH

Дальше. Что есть в Настройке Автоплатежа? Поля для ввода данных кредитки? Или у пользователя уже сохранены эти данные? Нужна ли поддержка АСН или других методов оплаты?

Уже для того, чтобы понять, что тут писать, требуется достаточно подробно обсудить возможности и требования.

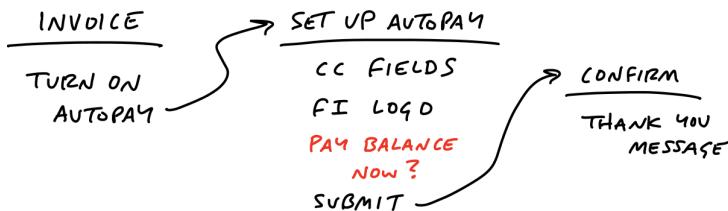
Пусть по мере этого обсуждения мы решаем спрашивать данные кредитки и показываем логотип финансового учреждения.



Вроде бы все просто. Но стоп! Отправка этой формы означает оплату изначального инвойса или нет? Хмм. Теперь у нас есть вопросы как к функционалу, так и к интерфейсу. Автоплатёж применяется только к будущим инвойсам, или к текущему тоже? В какой момент нам нужно объяснить это пользователю? Вопросы становятся глубже, а ведь мы написали всего несколько слов.

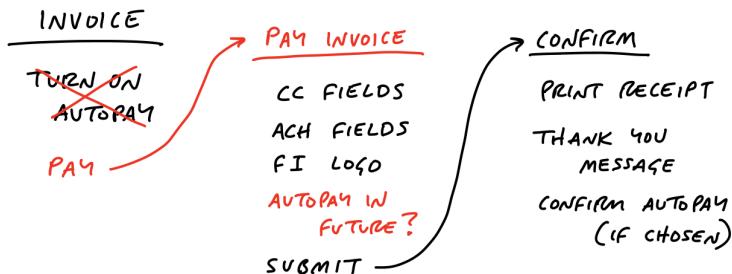
Лёгкость нашего наброска позволяет нам мгновенно оценивать разные варианты.

Мы можем добавить опцию в Настройки Автоплатежа...



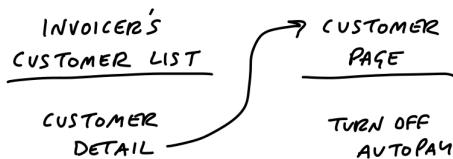
Но тогда экран подтверждения становится излишне сложным.  
Нужно показывать квитанцию и другую важную информацию  
об оплате.

Как насчёт другой идеи? Вообще не начинать со страницы  
Инвойса. Вместо этого, добавим опцию «Использовать  
Автоплатёж в будущем» в место Оплаты инвойса. Тогда будет  
ясно, относится ли Автоплатёж к текущему инвойсу. Можно  
добавить сообщение «Автоплатёж был подключен» в место  
Подтверждения оплаты.



Пока рисовали, вспомнили, что текущая форма оплаты  
поддерживает не только кредитки, но и АЧН. Быстремько  
обсуждаем, решаем тоже поддерживать АЧН.

Что дальше? Как отключить подключённый Автоплатёж?  
Сейчас у клиентов нет логинов и паролей, инвойсы оплачиваются через одноразовые ссылки. Сразу хочется добавить в проект разработку логинов и паролей. Однако, команда решает, что это слишком большой объем работы. Исходя из того, что мы знаем про клиентов, мы решаем, что будет достаточно, если человек, оплативший инвойс, будет обращаться к отправителю и просить отключить Автоплатёж.



Этот пример показывает, с какой скоростью и какой глубины идёт обсуждение макетной платы. Описывая «места», «действия» и «связи», мы можем быстро проработать принципиальные дизайнерские решения, не отвлекаясь на мелочи.

В конце этого процесса — когда схема кажется нам окончательной — в нашем распоряжении есть все элементы для того, чтобы сформировать проект.

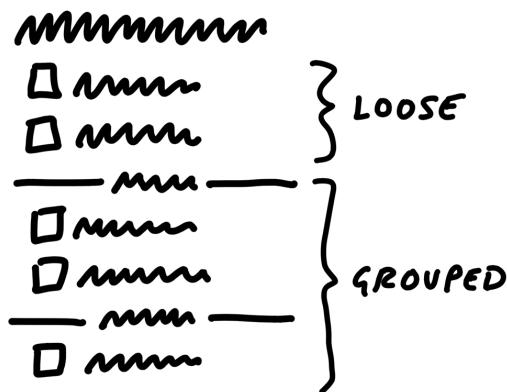
## Наброски толстым маркером

Иногда для обсуждения идеи принципиален внешний вид. Тогда набросок в стиле «макетной платы» не имеет смысла. Например, нужно определить именно расположение элементов (которое мы игнорируем, рисуя схему). Однако, мы по-прежнему не хотим тратить время на прорисовку полноценных

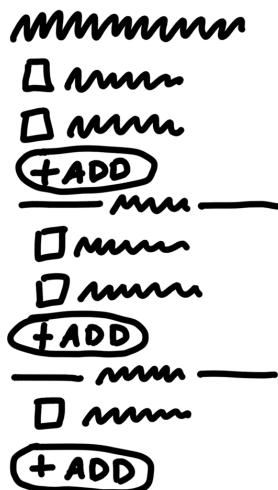
экранов, неизбежно включающих в себя кучу мелочей. Вместо этого, мы рисуем «наброски толстым маркером».

Смысль этого приёма — использовать настолько большой размер маркера, чтобы добавить мелкие детали было практически невозможно. Мы рисуем как настоящими маркерами на бумаге, так и на планшетах, кистью большого размера.

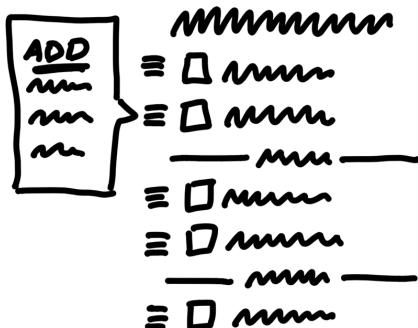
Вот пример. Мы заметили, что часто создаём «бесполезные» пункты в списках дел в качестве разделителей, например, «----». Появилась идея добавить поддержку «настоящих» разделителей, и нужно было изучить последствия этого решения. Допустим, в списке появляется разделитель, и тогда сверху будут «свободные» пункты, а под каждым разделителем — «сгруппированные».



В каждую группу (включая группу «свободных» пунктов) можно добавить пункт.



Не очень хорошо, что кнопки разбивают визуальное единство списка. Возможное решение — поместить кнопку «Добавить» внутрь меню, которое уже и так есть у каждого пункта.



Рисуя наброски, мы меньше ограничены, чем рисуя схемы. Поэтому надо следить за собой и не увлекаться мелочами, чтобы

случайно не получился полноценный экран, нарисованный жутко жирными линиями.

Может показаться странным, что мы выделяем наброски толстыми маркерами в отдельный инструмент при создании проекта. Но они позволяют нам «осмотреться вокруг», прежде чем погружаться в детали, и убедиться, что мы не пропустили ничего принципиального.

## Результат работы — элементы

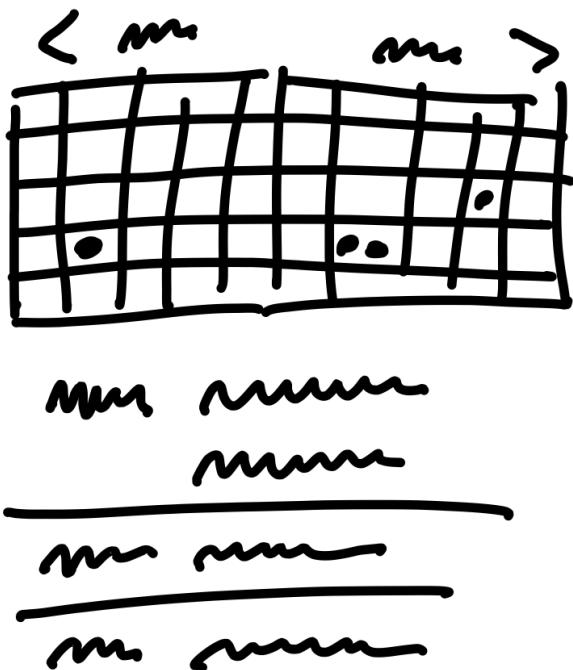
В примере с Автоплатежом, мы получили чёткий набор элементов:

- новая галочка «Autopay in future?» на экране «Оплатить инвойс»
- опция «отменить автоплатеж»

В примере с группировкой пунктов в списках задач, мы получили следующие элементы:

- «свободные» пункты выше первого разделителя
- сгруппированные пункты ниже разделителя
- кнопка «добавить» либо под каждой группой, либо в контекстом меню.

Похожим образом, мы набросали толстым маркером решение для календаря с точками:



Набросок позволил нам определить элементы:

- сетка на 2 месяца
- точки обозначают события
- под сеткой — простой список событий, который скроллится по мере нажатия на точки.

Ровно тот объем конкретики, который нужен нам для формирования проекта.

## **Свобода для дизайнера**

Когда придёт время передавать проект дизайнеру, не придётся говорить «Я нарисовал, но ты так не делай». Любые решения в ваших набросках влияют на решения того, кто с ними работает, даже если вы этого не планировали. Кроме того, набросок, который не говорит лишнего, позволяет всем двигаться быстро и оставляет свободное пространство для работы дизайнера.

В этом — суть формирования проекта. Мы отвечаем на все принципиальные вопросы, но оставляем место для будущих конкретных решений. Сформированный проект — это не «требования» и не «спецификации». Скорее, вы задаёте правила игры, а команда разработки по ним играет.

## **Ещё не готово к передаче**

Мы начали с расплывчатой идеи — например, «сгруппировать пункты в списке» — а сейчас у нас уже есть набор элементов решения. Но проект ещё не готов к передаче. Мы определили подход, он кажется правильным — теперь нужно проверить риски. Не осталось ли скрытых принципиальных вопросов? Не возникнет ли препятствий, которые помешают разработать проект в срок?

В следующей главе мы займёмся управлением рисками. А затем упакуем сформированный проект в презентацию.

## **Ещё не обязательство**

Также не забывайте, что на этом этапе от проекта вполне можно отказаться, по любым причинам. Пока не выделены никакие ресурсы, не взята ответственность. Мы просто изучили сырую идею и приблизили её к состоянию проекта, готового к передаче.

## Риски и белые пятна

---



Мы формируем проект, рассчитанный на конкретный срок (в нашем случае, 6 недель). В прошлой главе, пользуясь опытом и интуицией, мы определили элементы, которые, как нам кажется, укладываются в этот срок. Однако достаточно будет одного «белого пятна», чтобы выйти из бюджета и провалить проект. Представьте, команда находит непредвиденную проблему, на решение которой уходит 2 недели — треть бюджета потрачена впустую!

А иногда такие проблемы вообще не имеют очевидного решения. Однажды мы взяли в работу проект по редизайну списка проектов на главной странице Basecamp. Мы решили, что дизайнер вполне разберётся, и не сформировали проект как следует. Но в процессе оказалось, что задача намного сложнее, чем мы думали вначале. Спустя 6 недель никто так

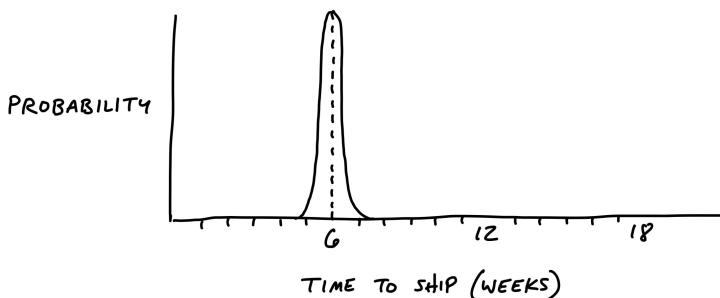
и не смог придумать подходящее решение, и проект был отменён.

Конечно, неопределённости будут всегда. Но это не значит, что не надо искать их как можно раньше. Цель — отдать в работу как можно менее дырявый проект.

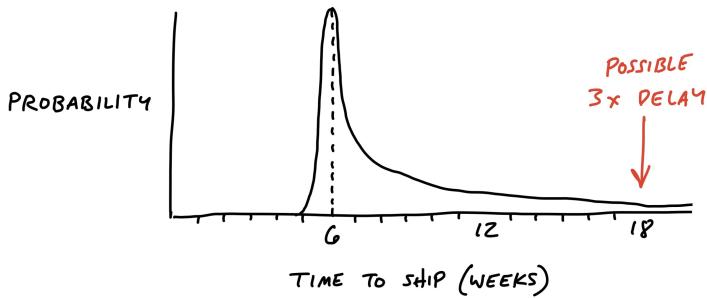
## Виды риска

С точки зрения рисков, хорошо сформированный проект выглядит как

распределение вероятностей с лёгким хвостом. Ключевые элементы понятны, белых пятен нет — проект будет завершён в срок плюс-минус неделя.



Однако любые белые пятна — технические вопросы, неверно истолкованные зависимости, и так далее — увеличивают вероятность задержки сдачи на порядок.



Чтобы минимизировать риск, мы описываем проект как набор независимых, понятных частей, которые соединяются понятными способами.

## Ищите белые пятна

В прошлой главе мы делали наброски элементов, и это была быстрая исследовательская работа. Она шла вширь, не вглубь.

Сейчас — время замедлиться и критично оценить выбранное решение. Мы ничего не упустили? Что может пойти не так?

Пройдите по сценариям решения в режиме «замедленной съёмки». Как конкретно пользователь попадает из начала сценария в конец? Подробный проход может выявить вопросы, требующие ответа.

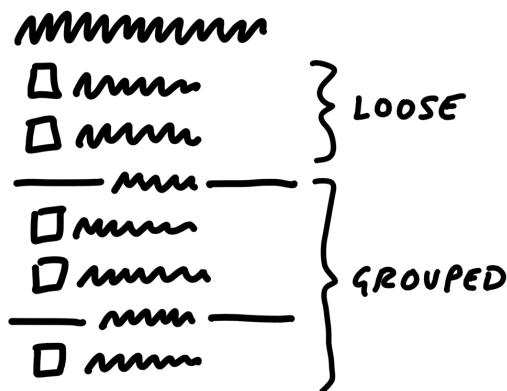
Трезво оцените жизнеспособность решения. Ищите ответы на вопросы:

- потребуется ли делать вид работ, который мы никогда раньше не делали?

- мы точно знаем, как соединяются части проекта, или только предполагаем?
- нужно ли сейчас пойти на какие-либо компромиссы, чтобы команда могла спокойно работать?

## Пример

Когда мы формировали проект «Группы в списках дел», мы добавили идею разделителей:

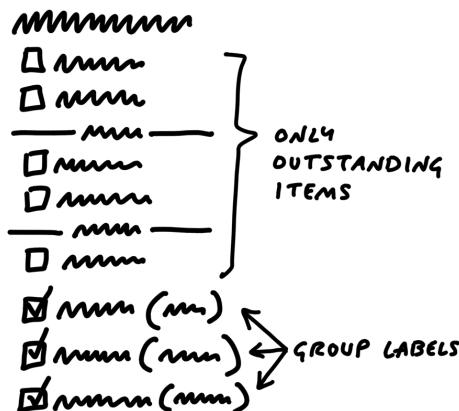


Сама идея выглядела хорошо, и логика «сгруппированных» и «свободных» дел как будто имела смысл. Но копнув глубже, мы обнаружили, что не знаем, как отображать завершённые дела. В конце каждой группы? Или в конце общего списка, повторив все разделители? Или вообще заново придумать, как мы работаем с завершёнными делами?

Это было белое пятно. Если отдать проект в работу в таком виде, на плечи команды ляжет несправедливая обязанность

доделать формирование проекта, а сроки при этом никто не отменял.

Из прошлого опыта мы знали, что любое изменение поведения завершённых дел влияет на сценарии использования, навигацию, эффективность, практически на всё. Чтобы исключить из проекта белое пятно, мы приняли решение оставить завершённые дела без изменений. Не группируем, не отделяем. Вместо этого, к каждому завершённому делу добавляем в скобках название группы. Это визуальный шум, но он оправдан — реализация стала кардинально проще.



Компромиссы такого рода трудно принимать, когда на тебя давит дедлайн. Можно было придумать множество других вариантов или вообще более глубоко переработать архитектуру списков дел. Дизайнер мог бы подумать — «возможно, если я поработаю над стилями, то завершённые дела внутри каждой группы окажутся лучшим решением». И потратить несколько дней (а иногда недель) часто на то, чтобы зайти в тупик.

Формируя проект, мы думаем не про внешний вид, а про свойства и риски. Приняв компромисс, мы не теряем никаких элементов найденного решения, и сильно уменьшаем риск.

Готовя презентацию, мы укажем на эту «заплатку», так что все, кого касается проект, поймут, что к чему.

## **Обозначьте границы**

Каждый в команде хочет сделать свою работу наилучшим образом. Поэтому все будут прорабатывать каждый вопрос максимально глубоко. Чтобы оставаться в рамках аппетита, пропишите, чего делать не нужно, какие сценарии не нужно обрабатывать.

Например, как-то мы работали над идеей уведомлений для групп пользователей. Вместо того, чтобы выбирать по одному пять разработчиков из длинного списка людей, хотелось просто нажать «разработчики». По мере обсуждения мы начали находить в продукте множество других мест, где такая логика была бы уместна. Почему бы не объединять людей в группы, когда добавляешь комментарий? Или когда создаёшь задачу? Или когда пишешь в чат?

Однако мы решили, что в рамках достижения конкретной цели необходимо ограничиться уведомлениями. Поэтому мы специально отметили, что остальные сценарии выходят за рамки проекта, и делать их не нужно.

## **Урезайте**

Возможно, в решении есть части, которые вызывают большой энтузиазм но не очень-то и нужны на самом деле. Когда мы формировали проект группировки в списках дел, мы очень хотели назначать группам разные цвета. Очевидно, выглядело бы очень круто, и даже немного более полезно. Но мы решили вырезать эту идею из презентации. Мы упомянули её как «было бы прикольно», но обозначили, что это не обязательная часть проекта.

## **Поговорите с техническими экспертами**

До этого момента, формирование проекта шло внутри вас (или вашей микро-группы из 2–3 человек). Прежде чем начинать работу над презентацией, определите, есть ли части проекта, реализацию которых вы не полностью понимаете. Возможно, вы исходите из предположений, которые лучше обсудить с экспертом или проверить на реальных данных.

Важно дать понять, что вы обсуждаете идею, а не принятый в работу проект. Ещё важнее не формулировать вопросы так: «Возможно ли это?» В мире разработке всё возможно, и ничто не бесплатно. Вам нужно узнать, реализуема ли идея в рамках вашего аппетита. «Возможно ли это за 6 недель?» — вот правильный вопрос.

Не просто спрашивайте эксперта «что ты думаешь об этом?». Ищите с его помощью белые пятна и риски провала.

Разговор с экспертом — ещё не презентация. Вместо документа или слайдов, достаточно разговора и набросков на доске или бумаге. Воссоздайте наброски найденного вами решения, проведите эксперта по сценариям. Только затем обсудите

замечания и изменения. Что может пойти не так? Что можно было сделать проще? А что стоит сделать иначе?

По результатам этого разговора (или нескольких) вы либо подтвердили свою правоту, либо возвращаетесь назад, пересматриваете идеи и формируете проект заново (целиком или частично).

## **Всё готово к презентации**

В конце этого процесса у вас есть:

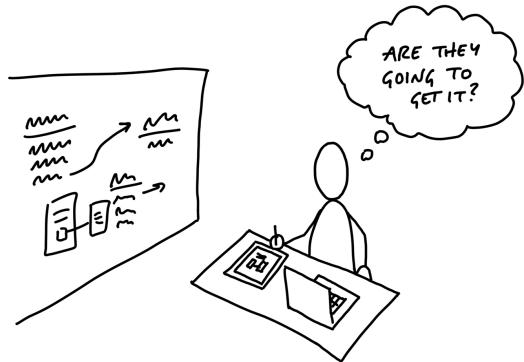
- набор элементов решения проблемы;
- варианты действий в местах потенциальных белых пятен;
- границы, обозначающие, чего делать не нужно.

От размытой идеи с непонятными перспективами мы перешли к ясному проекту с проработанными рисками, который не страшно представить команде.

Время создать **презентацию**. Её цель — сделать так, чтобы люди, не погруженные в контекст вашего проекта, поняли, какую проблему вы решаете, почему это важно, каково ваше решение, из чего оно состоит, а что делать не нужно.

# Презентация

---



Вы уверены в своей идее, она достаточно проработана для того, чтобы рассказать о ней широкому кругу людей. Но проект всё ещё в вашей голове, а также в непонятных набросках. Нужно придать проекту форму, с помощью которой другие люди смогут понять идею, оценить её, дать обратную связь и принять осознанное решение.

В этой главе мы изучим, из чего состоит презентация, и рассмотрим презентации реальных проектов Basecamp.

В наших презентациях всегда есть 5 ингредиентов:

- **Проблема** — сырая идея, сценарий, или что-то, что мотивирует нас работать над проектом;
- **Аппетит** — сколько мы хотим потратить и как это ограничивает решение;
- **Решение** — набор элементов, представленных в виде, в котором широкому кругу людей легко их понять;

- **Белые пятна** — части решения, которые стоит обсудить, чтобы избежать проблем в дальнейшей работе;
  - **Границы** — всё, что не нужно делать в рамках проекта.
- 

## Ингредиент 1 — проблема

Важно всегда презентовать проблему и решение вместе. Звучит очевидно, но удивительно часто презентации (включая наши собственные) начинались сразу с описания решения. Кажется, что проблема, которую проект призван решить, очевидна.

Начинать с «давайте сделаем» опасно. Не поняв хорошо проблему, нельзя понять, насколько подходит предложенное решение. Обсуждение проблемы также помогает понять друг друга позднее, во время принятия решения по передаче проекта в работу. Что, если решение отличное, но проблема актуальна только для узкого круга пользователей, которые не представляют реальной ценности для бизнеса? Уточнение проблемы перед обсуждением решения экономит время позже.

Насколько подробно описывать проблему — зависит от того, насколько ваши собеседники погружены в контекст. Обычно лучшее описание — это одна конкретная история, показывающая, почему сейчас не так, как надо. Команде будет легче примерить ваш проект на эту историю, чтобы оценить, насколько станет лучше.

## Ингредиент 2 — аппетит

Определение аппетита — продолжение определения проблемы. Мы хотим решить проблему не «вообще», а за 2 или 6 недель

силой одной команды.

Упоминание аппетита в презентации предотвращает непродуктивные обсуждения лучших решений. Лучшие решения есть всегда, если не помнить об ограничениях. Легко предлагать сложные и дорогие решения. А вот проработать простое решение с учётом аппетита уже не так просто.

## **Ингредиент 3 – решение**

Иногда на обсуждение попадают проблемы без решения. «Нужно сделать так, чтобы находить информацию в сообщениях было легче. Клиенты жалуются». Это ещё не проект, готовый к презентации. Отдать его в работу означает заставить команду вместо разработки заниматься исследованием.

В таких случаях необходимо вернуться к формированию проекта до момента, когда ясны проблема, аппетит и решение.

### **Помогите им увидеть**

Когда вы искали элементы решения, было важно найти нужный уровень абстракции — не делать слишком детальные наброски, чтобы не терять темп.

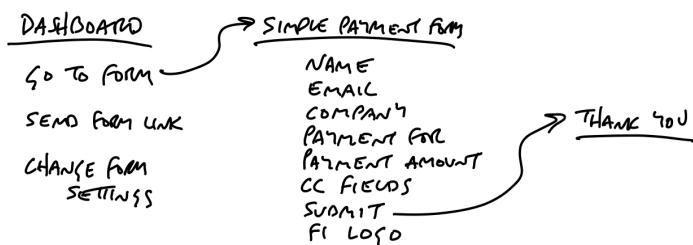
На этапе презентации задача немного другая. Решение уже найдено — нужно замедлиться и основательно подготовиться. Излишние детали по-прежнему не нужны, однако нужно удостовериться, что информации достаточно, чтобы люди со стороны могли понять и в чем проблема, и в чем решение.

То есть, конкретика нужна, но только чтобы достаточно ясно объяснить проблему и решение, а не нарисовать полноценные интерфейсы. Иначе есть риск увести обсуждение в цвета и шрифты.

К сожалению, наброски часто отражают ход ваших мыслей и ясны вам, но неясны другим. Если собеседник не формировал проект вместе с вами, разобраться в набросках может быть невозможно. Значит, нужны приёмы для презентации идеи широкому кругу людей без ухода в лишние детали. Вот эти приёмы:

### **Наброски поверх скриншотов**

Допустим, на этапе формирования вы набросали такую макетную плату:



Тем, кто в первый раз посмотрит на эту схему, трудно визуализировать эти элементы. Вы уже знаете, как выглядит экран, который вы хотите изменить, и представляете, какие изменения хотите сделать. Возьмите скриншот экрана, и сделайте «набросок толстым маркером» поверх:

Sally Conwell  
Pup Walkers Inc.

- Dashboard
- Invoices
- Pay Bills
- Accounting
- Reports
- Connected Apps
- Settings

**INVOICES**

Category	Amount
Coming Due	\$5,670
Past Due	\$3,200

**BILLS**

Category	Amount
Coming Due	\$567
Past Due	\$320

**Total Cash**

Account Type	Balance
Checking Account (*5637)	\$11,000
Savings Account (*4928)	\$5,000

**Accounting**

Metric	Value
Cash Flow MTD	\$560
Net Income YTD	\$24,000

**Monthly Invoice Summary**

Metric	Value
Total amount of sent invoices	\$2,700
Total amount of paid invoices	\$1,350
Average days to receive invoice payments	62

[VIEW INVOICES](#)

Хмм, всё ещё слишком много приходится додумывать.  
Добавить деталей в этом случае — правильное решение.

Your Payment Form  
See my Link to Customer Help!!!  
[CHANGE YOUR FORM SETTINGS](#)

Go To Your Form

**INVOICES**

Category	Amount
Coming Due	\$5,670
Past Due	\$3,200

**BILLS**

Category	Amount
Coming Due	\$567
Past Due	\$320

**Total Cash**

Account Type	Balance
Checking Account (*5637)	\$11,000
Savings Account (*4928)	\$5,000

**Accounting**

Metric	Value
Cash Flow MTD	\$560
Net Income YTD	\$24,000

**Monthly Invoice Summary**

Metric	Value
Total amount of sent invoices	\$2,700
Total amount of paid invoices	\$1,350
Average days to receive invoice payments	62

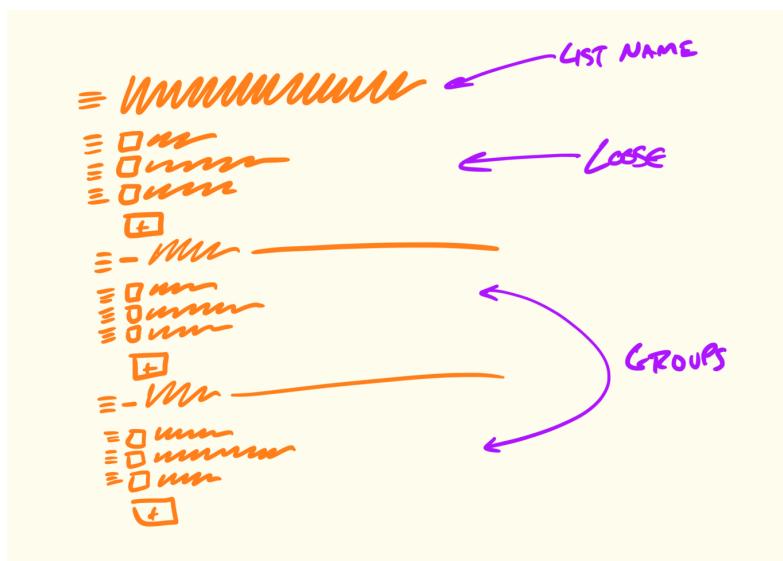
Взглянув на изображение, легче понять, о чём речь. Увы, нам пришлось принять некоторые дизайнерские решения по композиции нового блока, которые лучше бы оставить

на ответственности дизайнера. Стоит добавить в презентацию комментарий, что тут у дизайнера свобода выбора.

В этом примере мы сознательно добавляем деталей, которые помогают другим людям быстрее понять проект и убедить их. Впрочем, часто для понимания достаточно более простых приёмов.

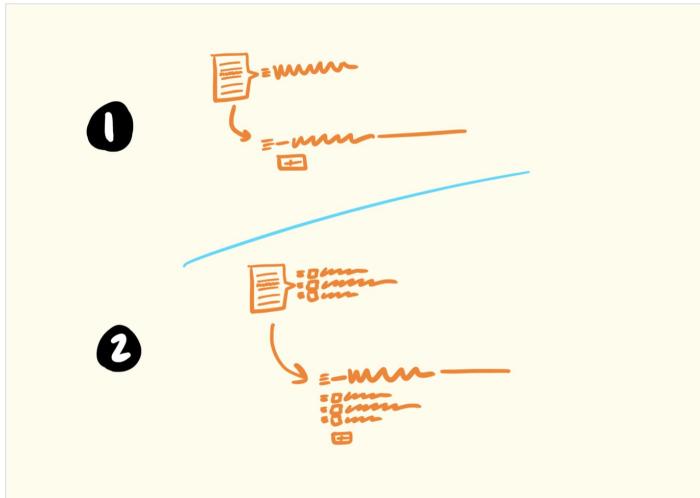
### Наброски толстым маркером с подписями

Для презентации некоторых идей наброски толстым маркером работают так же хорошо, как и ранее при формировании проекта. Стоит только сделать их более читаемыми, добавив подписи, например, так:



Или обозначения, которые помогут обращаться к конкретным элементам наброска в обсуждении:

You can create a divider in two ways...



Two ways to make a divider - 152 KB

## Ингредиент 4 – белые пятна

Для описания белого пятна часто достаточно нескольких строк текста. Например, в проекте добавления платёжной формы автор проекта обозначил как белое пятно вопрос создания URL-ов, которые в первой версии не могли быть произвольными. Подобного рода вещи не критичны для проекта в целом, но упоминать их полезно, чтобы обозначить неопределённости.

## Ингредиент 5 – границы

Имеет смысл отдельно указать, что не входит в рамки проекта. В примере выше, команда заранее решила, что поле ввода текста не должно поддерживать никакое форматирование

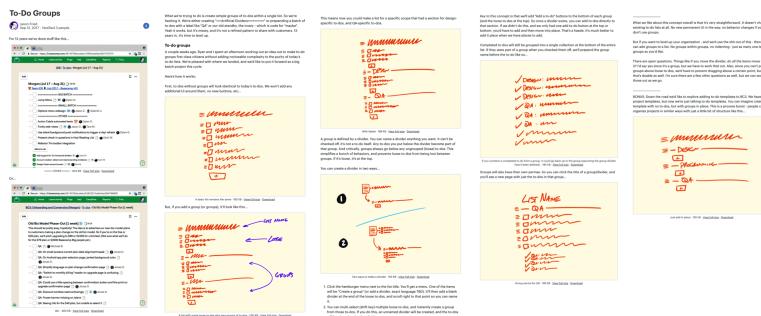
(WYSIWYG). Достаточно дать возможность загрузить логотип и ввести текст заголовка, для этого сделали отдельную страницу редактирования. Текст с форматированием — это то самое «лучшее» решение, которое было важно урезать с учётом аппетита проекта.

## Примеры

Вот пара реальных презентаций.

Первый пример — группировка пунктов в списке задач.

Презентация начинается с того, как пользователи в текущей версии продукта создают «костыльные» пункты-разделители. Затем идёт описание основных идей с набросками-иллюстрациями.



[Увеличить изображение](#)

Пара скриншотов демонстрируют проблему. Наброски толстым маркером демонстрируют решение и потенциальные белые пятна.

Второй пример — проект по изменению уведомлений.

**Group Notifications**

Philippe F. from Brain Design  
Last Tuesday I invited a bunch of folks for a meeting Groups are B2C to B2B ... Groups - Product

After looking at the pitch, I saw... pointed out it was extremely clear where to expand Groups to better. Therefore, we could accomplish a few other things with Groups. I think the main idea is to have a better way to manage groups and to have them be more dynamic.

he could present "new" and "existing Groups everywhere we can see all of that increases the scope. And it's a bad product design trend to add functionality for "something" that's not there.

With our new RCS module experiencing a need to scale and better manage the needs of our clients, we are looking for ways to make Groups more dynamic and flexible. We are here and how Groups would support us in this direction.

**Not enough detail on problem area**

For a group of people, it's important to know who is in the group. Who is the responsible, who is in charge, who is in a position to make decisions. It's a long time.

I asked you to demonstrate some workflows that already bring value that are part of what we do. I think that's a good place to start. I think that's a good place to start.

The last point really good one. One of the main reasons something fails. So probably 10% of the time it's because of a lack of communication between people or no communication.

**Message Board**

After I looked over the pitch and a similar pitch involving messaging, I could see the potential of this feature. It's something that we've been talking about for a while now.

It's not the kind of thing where mentioning everybody in a group to the team, but it's the kind of thing where you want to mention a specific person in a group without having to tag them individually.

With that in mind, I included design solutions for those two cases only.

**Guidelines for choosing Message notifications**

After I looked over the pitch and a similar pitch involving messaging, we have a single model today for grouping users. We can either use a simple group or a complex group. We can either use a simple group or a complex group. We can either use a simple group or a complex group.

With that in mind, I included design solutions for those two cases only.

**Do Events get a different solution?**

Going to the Events under "Find more" is not an straightforward. The current UI is a single search field.

**Workaround: This is only about Subscribers**

I came back to discuss with a colleague if it's hard to stay using the "Work Field" and instead relied on the "Who" field to be the "real field". It's a good opportunity to be more generic with the "Who" field. It's a good opportunity to be more generic with the "Who" field.

The word "would" mean that it's not for others. It's a good opportunity to be more generic with the "Who" field. It's a good opportunity to be more generic with the "Who" field.

The suggestion is we can refine the Groups feature entirely to "Who" should be modified? - The suggestion is we can refine the Groups feature entirely to "Who" should be modified? -

**Prepared solution**

We could define these "Subscription group" in AdminPanel, so planned protocols

**What we're not doing**

At a minimum, we can't implement the "Who" field. It's a good opportunity to be more generic with the "Who" field.

If we want to, we can implement it to allow regular people to interact with the service approach. The suggestion is we can refine the Groups feature entirely to "Who" should be modified? -

Last chance for this one is the next update. We'll shape up it down enough.

## Увеличить изображение

Два видеоролика демонстрируют проблему. Наброски толстым маркером и макетная плата — решение. Чёрные картинки ближе к концу — графики с данными, подтверждающими решение.

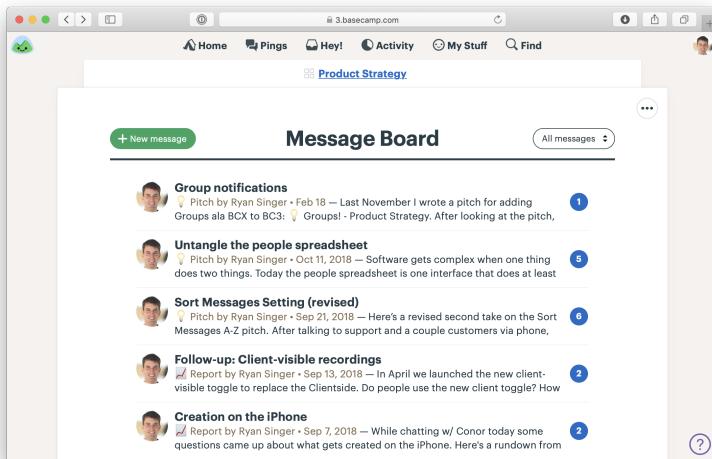
## Публикация и обсуждение

Мы предпочитаем асинхронную (письменную) коммуникацию и собираем встречу только если необходимо.

Таким образом, первый шаг — публикация презентации для всех участников процесса. У каждого будет возможность ознакомиться с документом заранее, в своём темпе, поэтому последующее обсуждение и голосование проходит быстро и продуктивно. Даже если кто-то из участников не успел просмотреть презентацию, вы сможете быстро пройти по готовому документу во время встречи.

## Как мы используем для этого Basecamp

Мы публикуем презентации в виде Сообщений в Basecamp, в категорию «Презентации». Доступ к презентациям есть у команды продуктовой стратегии. Они будут участвовать в обсуждении и голосовании.



Список презентаций в команде продуктовой стратегии в Basecamp.

The screenshot shows a web browser window for 3.basecamp.com. The navigation bar includes Home, Pings, Hey!, Activity, My Stuff, and Find. Below the navigation is a breadcrumb trail: Product Strategy > Message Board. The main content area has a title "Sort Messages Setting (revised)". A post by Ryan Singer from Sep 21, 2018, is shown, mentioning a "Sort Messages A-Z" pitch. The post discusses the challenge of managing many messages and proposes sorting by latest comment. It includes sections for "Example case", "Looking for a 1-weeker", and "A tool setting". A note at the bottom states that the tool setting from the original pitch is still the way to go. There are 6 notifications indicated in the top right corner.

## Sort Messages Setting (revised)

Pitch by Ryan Singer  
Sep 21, 2018 · Notified 4 people

Here's a revised second take on the [Sort Messages A-Z](#) pitch.

After talking to support and a couple customers via phone, the biggest immediate win would be offering to [sort by latest comment](#).

**Example case**

One customer described the problem very well. Her team posts work for clients to review on the Message Board. Sometimes a client is slow to respond. By the time the client responds to a proof, the thread might be 10 items down on the Messages list. This makes it nearly impossible find later when she's trying to determine where the project left off.

Did she try looking at Latest Activity? She said yes, but the problem is Latest Activity has too much stuff on it. All the events about To-Dos and other discussion drown out the feed.

Sorting Messages by latest comment would make it easier to see where the latest response was.

**Looking for a 1-weeker**

There might be a dozen different ways to improve this, but it's the kind of thing we'd like to solve in a very short iteration. We could add a sort setting and improve this for a lot of customers very quickly.

**A tool setting**

I think the tool setting from the original pitch is still the way to go here. This ensures that

Сообщение с презентацией. Обратите внимание на размер аппетита — всего 1 неделя.

Иллюстрации (например, наброски толстым маркером) мы обычно рисуем на iPad и вставляем внутрь сообщения как скриншот с подписью.

With that in mind, I looked for broad design solutions for those two use cases only.

**Solution for choosing Message subscribers**

We could use a BCX-style approach to solve the first case. We have a single modal today for setting Subscribers app-wide, whether you are posting a new Message or changing who gets notified about some commentable. That "Who should be notified?" modal could offer Groups at the top. Clicking them checks members of the Groups below.

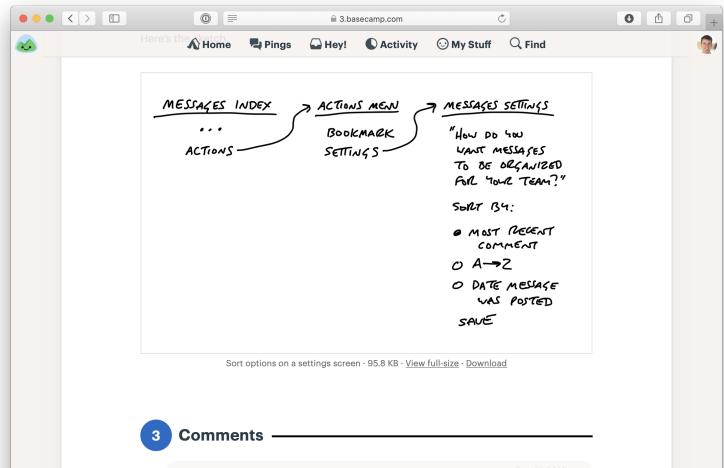


Groups on the Who Should Be Notified? modal - 90.7 KB · [View full-size](#) · [Download](#)

**Do Events get a different solution?**

Solving the Events "with" field case is not as straightforward. The current UI is a single autocomplete field.

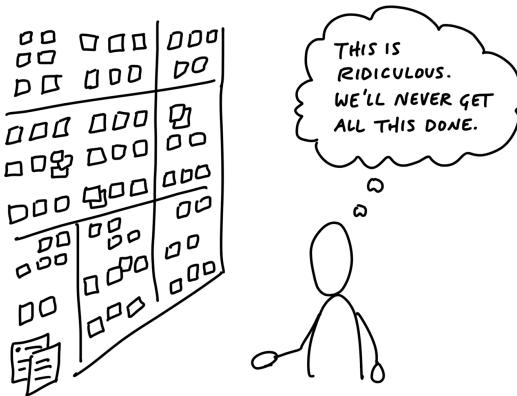
Участники оставляют комментарии асинхронно, по мере прочтения. Комментарии — не для принятия решений (для этого позже будет голосование), а для замечаний и ответов на вопросы.



В следующей главе мы рассмотрим, как презентации превращаются в запланированные проекты.

# Ставки вместо бэклогов

---



У нас готова презентация, что с ней происходит дальше?  
Прежде всего, она не попадает в очередь задач (бэклог).

## Никаких бэклогов

Мы считаем бэклог лишним грузом. В нём копятся десятки (иногда сотни) задач, на которые никогда не найдётся достаточно времени. Растущий бэклог создаёт ложное ощущение, что команды не успевают, даже когда это фактически не так. Не имеет смысла снова и снова натыкаться на задачу только потому, что несколько месяцев назад она показалась кому-то важной.

Бэклоги также сжирают невероятное количество времени.  
Вместо того, чтобы идти вперёд, посвящая время актуальным делам, вам приходится постоянно приводить в порядок устаревающие задачи.

## **Голосование**

Если не бэклог, то что? Перед началом каждого 6-недельного цикла мы встречаемся и проводим **голосование**, на котором принимаем решения о том, чем заниматься в этом цикле. В обсуждение и голосование берутся презентации, созданные или переработанные в течение предыдущего цикла.

Больше ничего не обсуждается. Никаких бесконечных списков идей. Только несколько презентаций — хорошо сформированных, с продуманными рисками. Авторы «делают ставки» — выносят свои презентации на голосование.

Встречи получаются нечастыми, короткими и невероятно продуктивными. Если за презентацию голосуют, она передаётся в работу на следующем цикле. Если нет — работа над ней прекращается. И всё, никакого управления задачами.

Что если презентация всем понравилась, но время брать её в работу по каким-то причинам неподходящее? В этом случае автор (или любой желающий) может выставить её на следующее голосование через 6 недель.

## **Независимые списки дел**

Если нет общего бэклога, неужели нужно держать всё в голове (а следовательно, всё время что-нибудь забывать)? Нет, каждый член команды ведёт учёт своих дел — презентаций, багов, запросов — так, как ему удобно, без централизации.

Например, команда поддержки может поддерживать собственный список типичных проблем. Продуктовая команда

ведёт список идей, из которых могут сформироваться новые проекты. У разработчиков есть список багов. Нет единого списка, и никакие из этих списков не влияют на голосование и передачу проектов в работу.

Регулярные (но нечастые) встречи разных команд помогают обменяться идеями по поводу предстоящих дел. Например, команда поддержки делится с командой продукта типичными запросами от пользователей. Возможно, команда продукта примет решение взять в работу только один запрос. Возможно также, на следующей встрече команда поддержки обратит внимание на другой важный запрос.

Таким образом у каждой команды есть независимость (а значит, ответственность) в управлении своими приоритетами. Люди из разных команд могут и должны доносить до других команд то, что считают важным — любым удобным способом.

Ещё один плюс — разговоры становятся более своевременными. Конкретный человек выносит на обсуждение тему, важную для него именно сейчас по определённой причине (а не просто «разобрать бэклог»).

## **Важные идеи никуда не денутся**

Очень легко переоценить идею, но будем честны, чаще всего идеи почти ничего не стоят. Они появляются отовсюду и накапливаются без перерыва.

То, что действительно важно, вернётся к вам снова и снова. Во-первых, вам будет труднее про это забыть. Во-вторых, даже

если это что-то не столь увлекательное (например, сообщение об ошибке), вам не дадут забыть (например, жалобы будут продолжаться). Если что-то всплыло в разговоре только однажды — вероятно, не такое уж это важное дело. А если вы продолжаете слышать об идее со всех сторон — явно стоит взять её в работу и сформировать проект.

# Голосование

---



Участники ознакомились со всеми презентациями, пришло время принимать решение — какие проекты брать в работу в следующий цикл.

## 6-недельные циклы

Трудно запланировать работу над будущим проектом, если неясно, у кого есть свободное время и сколько этого времени вам доступно. Обычно, если разные проекты пересекаются по времени, планирование командной работы становится похожим на бесконечный тетрис в календаре.

Работа 6-недельными циклами радикально упрощает планирование. Цикл задаёт нам стандартный размер проекта — сначала для формирования, затем для разработки.

Некоторые компании используют двухнедельный цикл («спринт»). Мы попробовали и решили, что две недели — слишком мало для того, чтобы реализовать что-то осмысленное. К тому же, в таких коротких циклах слишком много времени уходит на ненужное планирование.

Мы попробовали увеличить длительность цикла. Мы хотели, чтобы в пределах одного цикла было возможно реализовать проект, пусть и небольшой, целиком — от начала до конца. В другой стороны, уже в начале работы команда должна «ощущать» лёгкое давление дедлайна, чтобы оперативно принимать решения по ходу работы и не прокрастинировать. После долгих поисков мы остановились на 6 неделях.

## Перерыв

Если бы циклы работы шли вплотную, один за другим, ни у кого не было времени на выдох и обдумывание будущих дел. Конец цикла — самое плохое время для планирования, все заняты завершением текущих задач.

Поэтому после каждого 6-недельного цикла мы делаем двухнедельный «перерыв». Во время перерыва мы не работаем над запланированными проектами. Люди встречаются, обсуждают всё, что накопилось, думают, что и как делать дальше. Программисты и дизайнеры вольны работать над чем хотят. После напряжённых шести недель они наслаждаются свободой в управлении своим временем — исследовать новые идеи, изучить новые технологии и приёмы. Перерыв также используется для исправления несрочных багов.

## Размеры команды и проекта

Помимо стандартного размера цикла, мы также стараемся придерживаться стандартного размера проекта и команды.

Команда одного проекта обычно включает в себя одного дизайнера и одного-двух программистов. Позже к ним присоединяется тестировщик.

В течение цикла такая команда работает либо над одним проектом стандартного размера, либо над набором из нескольких мелких проектов. Размер мелкого проекта — обычно 1 или 2 недели. Мелкие проекты внутри набора не планируются заранее — команда работает над ними в удобном им порядке. Главное, чтобы все они были готовы к концу цикла.

## Процесс голосования

Голосование — это встреча во время двухнедельного перерыва, когда команда, отвечающая за стратегию, решает, что взять в следующий цикл. На обсуждение выносятся новые презентации, а также иногда пара более ранних презентаций, которые автор доработал и снова предлагает взять в работу.

Как я отметил ранее, никакого разбора очереди задач. Только несколько продуманных презентаций.

В Basecamp голосующая команда состоит из CEO (в нашем случае за ним последнее слово в продуктовых вопросах), СТО, ведущий разработчик и продуктовый стратег (это я).

У этих людей обычно дефицит времени, поэтому встреча проходит бодро и редко занимает больше часа или двух. У всех участников была возможность заранее просмотреть все

презентации. Многие вопросы обычно проясняются в комментариях или быстрых встречах ещё до голосования. На самом голосовании остаётся только принять решение.

Все участники голосования в курсе, какие люди будут доступны для работы в грядущем цикле, какие приоритеты у бизнеса, чем компания занимается в данный момент. Результат встречи — запланированный цикл. (Подробнее о процессе планирования — ниже).

Участвуют все руководители компании, поэтому решения голосования не требуют подтверждения, и никто не может задним числом их поменять или вмешаться в планирование.

Такой формат — встречи руководителей — очень важен для успешной работы цикла. Встреча не длится слишком долго, все презентации хорошо подготовлены, число принимающих решения невелико. Участники определяют направление развития продукта, а не спорят за ресурсы команд или приоритезируют задачи.

## **Голосование вместо планирования**

Мы используем термин «голосование», а не «планирование», и вот в чём различия.

Во-первых, отдать голос означает ожидать результат. Мы не просто заполняем календари команд, пока они не заполнятся. Мы не просто распределяем задачи по неделям, надеясь, что завершение задачи улучшит продукт. Мы выдаём команде кредит в 6 недель, ожидая конкретного результата, в важность которого мы поверили.

Во-вторых, отдать голос означать передать полномочия.

Выданные команде 6 недель означают, что она работает над проектом автономно, не отвлекаясь ни на что другое.

Мы не пытаемся контролировать каждый час каждого разработчика.

В-третьих, отдать голос не означает взять слишком большой риск. Максимум, что можно потерять — 6 недель.

Мы не позволяем себе оказаться в ситуации, когда на неудачный проект потрачено на порядок больше времени.

Рассмотрим подробнее два последних пункта.

## **Не отвлекать**

Несправедливо выдать команде 6 недель на реализацию проекта, в процессе выдавая им ещё какие-то задачи.

Мы не позволяем никому отвлекать команду в процессе работы над проектом. Каждая незапланированная задача означала бы, что мы нарушаем договор, который мы заключили с командой — их 6 недель.

Не стоит верить «тут быстренько» или «мне всего пару часов».

Для достижения результата требуется непрерывная последовательность действий. Когда вы отвлекаете кого-то на день, теряется не только календарный день, а всё «непрерывное ускорение», накопленное им к этому моменту. Отвлечение на день может стоить проекту недели.

Что если в течение цикла появляется информация, требующая внести изменения в проект? Мы по-прежнему не прерываем команду и не нарушаем договор. В худшем случае мы потеряем

6 недель прежде, чем начнём реагировать на полученную информацию в конце цикла, на очередном формировании проектов и голосовании.

Вот почему так важно планировать только один ближайший цикл. Это позволяет нам достаточно оперативно реагировать на ситуацию. Конечно, в случае настоящего ЧП мы дёргаем стоп-кран. Но настоящие ЧП очень редки.

## Предохранитель

Мы сочетаем правило «не отвлекать» с ещё одним жёстким, но очень эффективным правилом. Команда обязуется реализовать проект к концу цикла. Если проект не завершён, по умолчанию он отменяется. Мы сознательно идём на риск не получить запланированный результат. Звучит сурово, но это очень полезно для всех, кто работает над проектом.

Во-первых, мы уменьшаем риск выхода проекта из-под контроля. Мы договорились о размере «аппетита», когда проект сформирован, перед началом работы. Было бы глупо позволять тратить на проект в два, три раза больше запланированного времени. Очень редки ситуации, когда результат нужен любой ценой.

Мы называем это правилом «предохранителя», который защищает всю систему от перегрузки одним проектом.

Во-вторых, если проект не успевает, чаще всего это означает просчёт в процессе формирования. Предохранитель заставляет нас вернуться к формулировке проблемы. Мы можем переосмыслить проект для нового цикла, или придумать новое

решение с учётом найденных белых пятен. Затем переформированный проект проходит через стандартное голосование, на котором решают, стоит ли он ещё 6 недель.

В-третьих, правило предохранителя напоминает команде, что она отвечает за проект. Как мы увидим в следующей главе, команды получают полную ответственность за выданный им проект. В том числе это значит полномочия решать, чем пожертвовать и какие углы срезать. Жёсткий дедлайн мотивирует команду регулярно проверять, как их решения влияют на объём оставшейся работы.

## Исправление багов

Если мы не отвлекаем разработчиков в течение цикла, как мы исправляем баги?

Прежде всего, давайте подвергнем сомнению общепринятое мнение о багах.

В багах нет ничего, что автоматически делает их важнее всей остальной работы. Тот факт, что нужно исправить баг, ещё не даёт повода прерывать свою или чужую работу. Баги есть во всех программах. Весь вопрос — насколько они критичны? Бывает, что теряются данные, продукт виснет, или пользователи массово видят что-то — тогда на исправление бросают все силы. Но подобные ЧП очень редки. Почти все баги могут дождаться конца цикла, а некоторые вообще не стоит исправлять. Если бы мы всегда отдавали приоритет всем багам, мы не смогли бы реализовать ничего нового.

При этом, конечно, никто не любит неисправленные баги. У нас есть три способа работы с ними не в ущерб проектам.

- 1. Период двухнедельного перерыва.** У каждого разработчика есть список вещей, которые они **хотели бы** исправить. Перерыв между циклами даёт эту возможность. Две недели через каждые шесть недель чаще всего достаточно.
- 2. Презентация и голосование.** Если размер исправления слишком велик, можно презентовать его вместе с другими проектами, соревнуясь за ресурсы команд. Допустим, какой-то серверный процесс замедляет работу продукта и разработчик хочет переписать его, сделав асинхронным. Он презентует проект, обозначив проблему, описав решение и белые пятна. Тогда, вместо того, чтобы прерывать работу над другими проектами, руководство может проголосовать и поставить проект в план будущего цикла.

Есть огромная разница между тем, чтобы исправлять баги, жертвуя запланированной работой, и делать исправление багов запланированной работой.

- 3. Багатон (марафон исправления багов).** Каждый год (обычно ближе к праздникам) мы посвящаем целый цикл исправлению багов. Мы называем это Багатон (bug smash). Люди как раз готовятся к праздникам, возможно переезжают или берут отпуска — неподходящее время для обычных проектов. Команды самостоятельно выбирают, какие баги они хотят исправить.

## С чистого листа

Ключ к успеху управления циклами — в том, чтобы начинать каждый цикл с чистого листа. Это значит, что в новый цикл не переносятся никакие остатки прошлого цикла. Каждый проект, взятый в цикл, должен пройти формирование, обсуждение и голосование.

Мы не знаем, что произойдёт в течение 6 недель — возможно, появится идея отличного проекта, или придёт очевидно критичный запрос.

Даже несмотря на то, что у нас есть какое-то подобие «дорожной карты» (стратегии развития), выходящей за рамки одного цикла, мы обсуждаем её отдельно, и не смешиваем с обсуждением проектов. Каждый цикл мы лучше понимаем, что работает, и что важно, а что нет. Планирование не дальше одного цикла даёт нам свободу реагировать на новую информацию.

Что если проект не может вместиться в 6 недель? Мы всё равно ограничиваем себя одним циклом. Допустим, некая фича займёт 2 цикла разработки. Мы формируем последовательные проекты, рассчитанные на 6 недель, таким образом, чтобы результатом каждого проекта к концу цикла было что-то работающее, доказавшее свою реализуемость.

Если первый цикл прошёл без больших проблем, мы с лёгким сердцем отдаём в новый цикл следующий проект. Но если проект забуксовал, у нас есть варианты. Можем переосмыслить идею в принципе. Можем поставить его на паузу и проголосовать за более срочные проекты. Принципиально то,

что мы всегда договариваемся о том, что должны получить в конце каждого цикла, и всегда можем сменить курс между циклами.

## **Делайте ваши ставки**

---

### **Изучите обстановку**

Многое в 6-недельном цикле зависит от того, посвящён ли проект развитию существующего продукта (фичи) или созданию нового.

Принимая решения, нужно хорошо понимать, в какой стадии развития находится продукт в целом.

### **Развитие продукта**

Добавляя фичи в существующий продукт, мы идём по стандартному процессу — формирование проекта, презентация, передача в работу. Мы ожидаем от команды разработки конкретного результата в конце цикла.

Весь уже существующий продукт, который *не изменится* в результате проекта — это окружение, оно известно заранее. Такие проекты подобны созданию мебели для уже готового дома.

### **Новые продукты**

Другое дело — новые продукты. Если, покупая диван, вы уже знаете размеры и обстановку комнаты, то разработка нового продукта — это скорее заливка фундамента и возведение стен.

Со временем мы выделили три этапа работы над новыми продуктами. На каждом этапе, процесс работы различается. Даже если каждый этап занимает несколько циклов, планирование и голосование происходит в рамках одного цикла.

## **Этап 1. Исследование**

В самом начале у нас есть только гипотеза. Мы не знаем, соединяются ли разрозненные идеи в один стройный продукт, ещё не продумана архитектура, и так далее.

Много работы уйдёт в корзину. На половине пути мы вполне можем решить, что нужно попробовать что-то совсем другое. То есть ещё нельзя качественно сформировать проект и сказать всем на презентации — «вот это именно то, чего мы хотим, и это можно получить за 6 недель».

Чтобы точно понять, чего мы хотим, придётся это создать. Мы называем этот этап «Исследование», и вот его отличия.

1. Вместо хорошо сформированного проекта, мы голосуем за время на разработку ключевых идей нового продукта. Проекта как такого нет — мы ожидаем, что поймём, что нужно, в процессе.
2. Вместо отдельной команды разработки, продукт берёт в работу руководство. Дэвид (СТО) — разработчик, Джейсон (CEO) — дизайнер. Это важно по двум причинам. Во-первых, нельзя делегировать другим работу, если не сможешь сформулировать, чего точно хочешь. Во-вторых, архитектурные решения, принятые на этом этапе, повлияют на будущее продукта в целом. Команда должна

опираться на своё видение продукта и отвечать за долгосрочные последствия решений.

3. Вместо реализованного проекта в конце цикла, мы ожидаем понимания. Цель — разобраться, а не разработать. В идеале у нас будет какой-то интерфейс и код, которые послужат основой для последующей разработки.

Пользователи не увидят никаких результатов в конце цикла исследования. Но мы всё равно планируем не дальше одного цикла. Возможно, к концу шестой недели мы поймём, что не готовы всерьёз браться за продукт. Или, напротив, прототип продукта сложится и будет готов к формированию проектов. Решение о продлении исследования мы принимаем в конце каждого цикла.

## Этап 2. Разработка

В какой-то момент все принципиальные архитектурные решения приняты, и основа для создания и развития продукта заложена. Теперь руководство может привлекать другие команды. Продукт переходит к этапу «Разработки», со стандартным процессом, описанным выше. Он идентичен работе с существующим продуктом. Основа, заложенная на этапе исследования, позволяет всем командам понять, что им нужно делать, и как результаты их работы соединяются в единый продукт.

1. Проекты снова проходят через полноценное формирование и презентацию. Все точно понимают, что должно получиться в конце цикла.

2. Над продуктом может параллельно работать несколько команд (каждая над своим проектом).
3. Цель работы — снова реализованный проект. Но поскольку продукт ещё не выпущен на рынок, «реализованный» в этом случае означает не «публичный релиз», а вливание кода в общую базу с намерением больше его не менять.

На этом этапе мы можем позволить себе экспериментировать — убрать уже готовую фичу из продукта, проголосовать за фичу, не будучи уверены, что она пойдёт в финальный продукт, и так далее.

### **Этап 3. Чистка**

Финальный этап перед выпуском продукта — «Чистка». Мы откладываем в сторону все процессы.

Мы выпустили достаточно продуктов, чтобы знать наверняка — всегда есть что-то, о чём все забыли, что-то пропустили, мелочи не на своём месте, ошибки, которые остались незамеченными. Волосы всегда встают дыбом, когда подносишь палец к кнопке «Опубликовать».

Поэтому мы всегда оставляем запас времени перед запуском. Вот чем отличается этап Чистки:

1. Проекты не формируются. Этот цикл скорее напоминает Багатон. Руководство оперативно определяет приоритеты и убирает отвлекающие факторы.
2. Нет чётких команд. Каждый берётся за то, с чем лучше всего может помочь.
3. Результаты заливаются в кодовую базу по мере готовности, не дожидаясь конца цикла.

Дисциплина при этом по-прежнему важна. Чистка не должна продолжаться дольше двух циклов. Это также последняя возможность для руководства вырезать фичи. Меньший размер первой версии означает меньшую нагрузку на поддержку пользователей и кода в будущем. Часто только когда видишь готовый к выпуску продукт, понимаешь, без чего можно обойтись.

## Примеры

### Календарь с точками

Календарь с точками был новой фичей для существующего продукта, Basecamp. Мы сформировали проект, выделили для него 6 недель, команда его разработала, и в конце цикла мы опубликовали его для пользователей. И рассказать особо не о чем :-)

### HEY

В 2020 году, после двух лет работы, мы запустили новый сервис электронной почты — [HEY](#).

В течение года продукт был на этапе Исследования. Команда из трёх человек (Джейсон, Дэвид и Джонас, дизайнер) проработала много очень разных идей, прежде чем прийти к пониманию будущего продукта.

Затем ещё год продукт был на этапе Разработки. Все команды принимали участие в создании набора фичей, составляющих HEY. После ещё двух циклов Чистки, во время которых мы удалили заметное число фичей, в июле 2020 года продукт был выпущен на рынок.

Строго говоря, был небольшой перехлёт этапов исследования и разработки. Большой размер компании позволил руководству передать часть продукта на этап разработки, продолжив исследование в других частях.

Каждый цикл на первом этапе исследования зависел от результатов предыдущего. Команда не планировала работать над продуктом именно два года. Уверенность в идеи продукта пришла постепенно, и в какой-то момент руководство решило, сколько именно циклов можно потратить на разработку (но при этом ещё не сформировало проекты для этих циклов). Мы всегда имели в виду возможность отложить работу над новым продуктом и вернуться к основному продукту, Basecamp.

## **Эксперимент: графики-холмы**

Третий пример — нечто среднее. Когда мы работали над идеей графиков-холмов в Basecamp (подробнее в главе 13), мы не были уверены, что идея вообще сработает. Basecamp был уже стабильным продуктом, и казалось рискованным отдавать пользователям экспериментальную фичу.

Мы решили организовать работу как будто находимся на этапе Разработки нового продукта. Сначала за один цикл мы сделали минимум, которым могли бы пользоваться сами.

Мы не планировали выпускать что-либо в свет в конце цикла. Если бы мы поняли, что идея не работает, потеряли бы этот цикл. Если бы появилась более важная задача, мы бы отложили эксперимент. Но в конце цикла появилась уверенность, мы сформировали проект как положено, проголосовали, отдали в работу на следующий цикл, и потом опубликовали.

# **О чём спрашивать на голосовании**

## **Стоит ли решать проблему?**

Как и в самой презентации, на обсуждении мы всегда отдельно говорим о самой проблеме. Не важно, насколько хорошо решение, если проблему не стоит решать.

Кажется, что любая проблема, которая затрагивает пользователей, имеет значение. Но проблем всегда больше, чем времени, чтобы их решить. В этом смысле голосования — выбрать, прямо сейчас вот эта проблема важнее, чем та и вон та?

Критерии решения у каждого свои, в зависимости от роли и опыта. Например, проблема может затрагивать малую долю пользователей, но чрезмерно нагружать поддержку. Члены команды, работающие с пользователями и поддержкой, оценят эту проблему по-разному.

Иногда сложность решения приводит к переосмыслинию проблемы. Точно ли нужно столько работы? Можно ли упростить формулировку, чтобы 80% проблемы решить за 20% усилий?

## **Адекватен ли аппетит?**

Допустим, участник встречи сомневается, стоит ли тратить 6 недель на обсуждаемый проект. Вот как может сложиться разговор:

1. Возможно, проблема не была сформулирована достаточно полно — автор проекта находит аргументы, склоняющие

чашу весов. Например, «да, такое происходит редко, но когда происходит, пользователи настолько расстроены, что это влияет на репутацию». Или, «звучит просто, но поддержке приходится каждый раз проходить через утомительную инструкцию в 11 шагов, чтобы решить проблему».

2. Стоит уточнить «а если ужать срок до 2 недель?».

Возможно, выяснится, что участника смущает не траты ресурсов сама по себе, а что-то в проекте. Например, «я понял, очень не хочется добавлять в эту часть продукта ещё одну зависимость».

3. Если автор проекта чувствует, что интереса у других участников встречи нет, можно просто положить проект на полку или отменить.

4. Также можно вернуться к этапу формирования и либо сделать мини-версию с меньшим аппетитом, либо глубже исследовать проблему, чтобы презентовать её решение более убедительно.

### **Устраивает ли решение?**

Проблема важна, аппетит адекватен, а всё ли хорошо с предлагаемым решением?

Например, каждый новый элемент интерфейса имеет цену: место на экране пользователя. Возможно, кнопка в углу главной страницы идеально решает проблему. Но заняв этот угол сейчас, мы не сможем использовать его в будущем. Оно точно стоит того?

Впрочем, обычно мы стараемся не искать ответов на вопросы дизайна и технологий на голосовании.

### **Подходящее ли время?**

Следующие проекты часто зависят от предыдущих. Возможно, мы слишком давно не выпускали новых фичей. Или наоборот, за время выпуска новых фичей накопились запросы на исправления и улучшения. Или команды слишком долго занимались чем-то одним, и стоит переключить их на другую область.

Эти аргументы вполне резонны, даже если обсуждаемый проект всех устраивает, но время для его реализации не самое подходящее.

### **Доступны ли нужные люди?**

В процессе голосования мы решаем, кто именно будет заниматься каждым проектом. У нас есть группа дизайнеров и группа разработчиков, и для каждого проекта перед каждым циклом мы собираем из них команду из одного дизайнера и одного-двух разработчиков. Эта команда работает вместе в течение всего цикла — а затем для другого проекта команды собираются заново.

Разные проекты требуют разных навыков. Также, у разных людей в разное время могут меняться предпочтения — например, кто-то долго работал над мелкими проектами и предпочёл бы заняться полноценным 6-недельным проектом.

Ну и конечно, всегда происходит небольшой Тетрис в календаре. Несмотря на стандартный размер проектов и циклов, бывают форс-мажоры, отпуска и переезды.

Есть компании, которые позволяют сотрудниками самим выбирать, над какими проектами работать, и этот процесс работает для них хорошо. Лично для нас такой способ не подходит.

## Опубликуйте сообщение с результатами

После голосования кто-нибудь из руководства пишет сообщение с результатами — какие проекты взяли в следующий цикл и кто над какими проектами работает.

The screenshot shows a Mac OS X desktop with a web browser window open to [3.basecamp.com](https://3.basecamp.com). The browser title bar says "3.basecamp.com". The page is a "Message Board" section of a project named "BC3". A post by Jason Fried is highlighted:

**Sill: The last cycle of 2017**

 FYI by Jason Fried  
Nov 17, 2017 - Notified 53 people

It's next cycle time!

First, this is a short cycle. We've got a few major holidays this cycle, plus the end of the year. People are busy with life, travel, snow (!), and all the other stuff that goes with ramping down one year and starting another.

So with that in mind, we're going to do something a bit different this cycle. We're only going to schedule a few core projects focused on making Basecamp feel simpler, a couple of experimental projects, and no dedicated small batch projects.

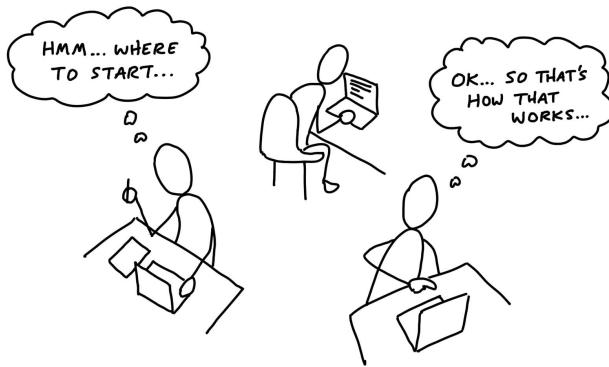
**Merge Latest Activity and Reports**  
Reports will move into the Latest Activity screen. When you click "Latest Activity" (or whatever we call it) in the nav bar, you'll see a series of report icons at the top, and "Latest Activity" will be the first one. So activity basically becomes a report like all the others, except that it's the first one you see automatically.

**Working prototype of interactive Hill Charts**  
We've been making Hill Charts manually in Apple Numbers and pasting them into Basecamp. Here's a [great example of charts mapping the search-in-place project](#) during last cycle.

This next cycle we're going to build an interactive prototype so we can do the mapping and dragging and positioning on a hill from right inside Basecamp. Scott is going to run the project and lead design. The goal is to have something we can use internally the cycle after next. If we continue to dig it, we'll release it as a full fledged feature and spend

# Передайте полномочия

---



Мы определились с проектами, пришло время нового цикла. С чего начать командам разработки?

## Проекты, а не задачи

Никто никому не назначает никаких задач. Нет «менеджера задач» или «архитектора», который декомпозирует проект на кусочки и раздаёт по кусочку разным людям.

Разделять проект на задачи заранее — всё равно что запихнуть презентацию в шредер. У каждого будет свой кусок. Наша цель — всегда иметь перед глазами проект целиком, чтобы у всех была общая картина.

Вместо этого мы выдаём команде проект целиком и ограничиваем их работу лишь тем, что описано в презентации. Команда сама распределяет работу и выбирает подход. У команды есть полная независимость в реализации

проекта так, как им удобнее. Этот метод отлично работает с талантливыми людьми, намного лучше, чем выдача тикетов в таск-трекере.

Обычно нельзя в начале цикла полностью описать, что именно нужно сделать для достижения результата. Любые заранее описанные планы придётся менять на ходу. И лучше всего поручить это самим дизайнерам и разработчикам.

Получая отдельную задачу, человек обычно не чувствует ответственности за то, как результат этой задачи соединяется с остальными частями. Планируя заранее, вы отгораживаете себя от реальности по ходу работы.

Мы не даём команде полную свободу. Мы сформировали проект, обозначили границы и уверены, что команда самостоятельно заполнит все детали, примет все дальнейшие решения, превратив проект в работающий продукт.

Вот почему так важно найти нужный уровень абстракции во время формирования проекта, без лишних деталей. Обладая талантом и опытом, команда разработки добьётся результата быстрее и лучше, чем если бы мы прописывали им максимально чёткое ТЗ.

## **Критерий готовности — деплой**

В конце цикла команда должна опубликовать на сервер результаты работы (*deploy*). Если в работе было несколько мелких проектов, команда может публиковать их по своему усмотрению до конца цикла.

Это требование позволяет работать правилу Предохранителя. Проект должен быть реализован за отведённое время, иначе наше голосование ничего не стоит.

Всё тестирование и проверка качества (QA) происходят **внутри** цикла. Команды добиваются этого, реализуя основные части проекта как можно раньше и передавая их на проверку по мере готовности (подробнее об этом позже).

Обычно мы не требуем работать в течение цикла над такими вещами, как документирование, маркетинг, описания для пользователей. У этих работ нет большого риска (они никогда не отнимают в 5 раз больше времени, чем ожидалось), их часто делают другие команды во время двухнедельного перерыва между циклами.

## Старт проекта

Формально проект начинается с создания проекта в Basecamp и добавления в него команды. Затем мы добавляем сформированную презентацию в Сообщения.

3.basecamp.com

Home Pings Hey! Activity My Stuff Find

BC3: Hill Charts > Message Board

## Hill Charts concept

Ryan Singer Nov 17, 2017 - No one was notified. 1

Here's the starting concept for the Hill Chart feature in Basecamp.

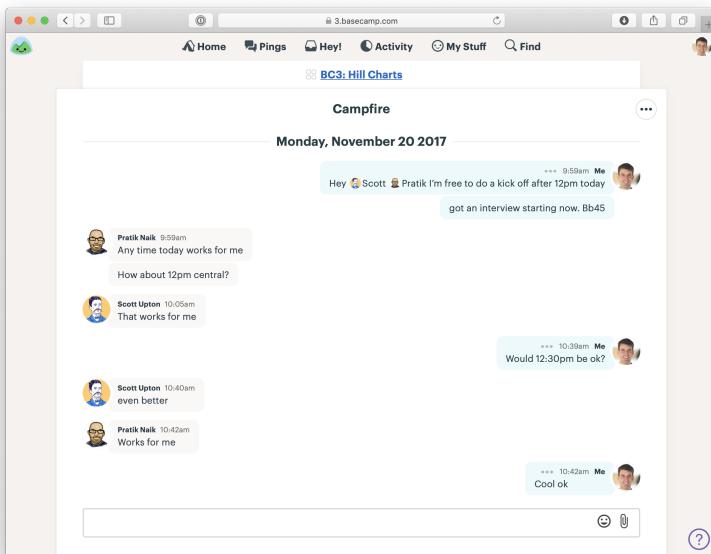
**How they relate to to-dos**

The basic idea is there's a hill chart at the top of the todoset. Hill **charts** map one-to-one with todosets, and the **dots** on the hill map one-to-one with todolists.

← HILL ON TOP

← TODO LISTS

Команды работают удалённо, поэтому мы используем чат в Basecamp, чтобы организовать звонок.

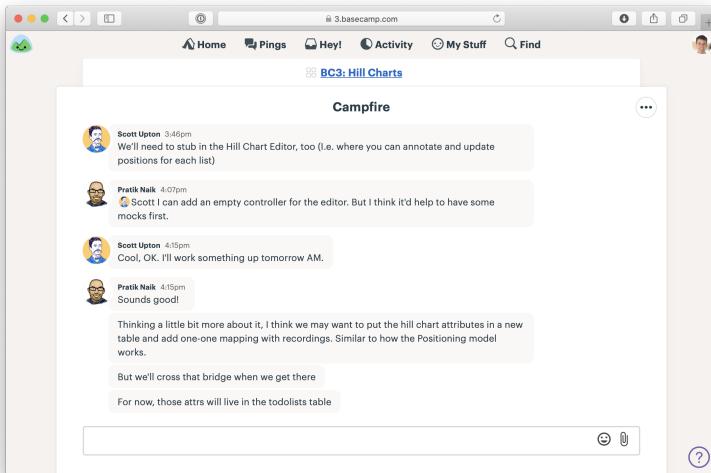


На встрече команда задаёт уточняющие вопросы.

## Утаптываем поляну

Первая пара дней работы над проектом не выглядит как настоящая работа. Никто не отмечает задачи выполненными. Никто не публикует результаты. Показывать никому нечего. Даже обсуждений почти нет.

Почему? Каждый член команды изучает ситуацию и обдумывает дальнейшие шаги.



Менеджерам важно уважать этот странный период. Нельзя требовать от команд сразу начать выдавать результат. На то, чтобы изучить существующее окружение, составить в голове план «путешествия» к результату, требуется время. Просить видимых результатов в это время бессмысленно. Намного лучше дать команде понять, что у них есть право спокойно подумать, прежде чем начать.

Обычно для периода молчания достаточно трёх дней, потом можно начать аккуратно вмешиваться.

## Придуманные и обнаруженные задачи

Поскольку команда получила проект целиком, задачи им нужно запланировать самим. Вначале команда ставит себе «придуманные» задачи — что, как им кажется в результате обдумывания, нужно начать делать.

А вот по мере того, как команда погружается в работу, возникают вещи, которые не были видны заранее. Из них и состоит основная работа над проектом, и чаще всего в них скрывается всё самое сложное.

Команда «обнаруживает» задачи по мере работы. Например, дизайнер добавляет кнопку в интерфейс, но потом обнаруживает, что в мобильной версии для неё нет очевидного места. Создаётся задача — разобраться, что делать с этим действием в мобильной версии.

Или первая итерация дизайна выглядит стройно и логично, но потом дизайнер обнаруживает, что нужно добавить пару абзацев объяснения, которое ломает всю сетку. Создаётся две задачи: переверстать экран с учётом нового текста, написать чистовой текст.

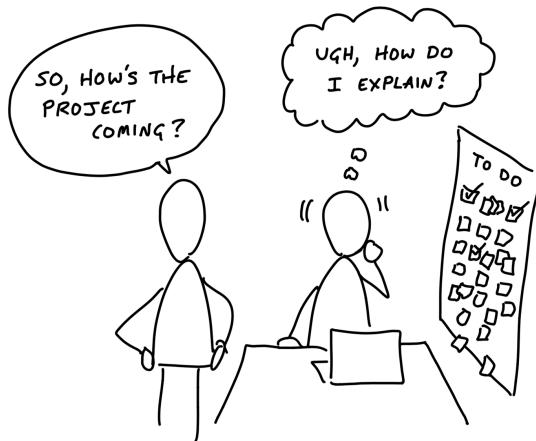
Часто задачи обнаруживаются в неожиданных местах. Допустим, разработчик занимается миграцией базы данных. Изучая взаимосвязи, она обнаруживает функцию, которую нужно обновить в другой части продукта. Она создаёт себе задачу обновить эту функцию позднее.

Итак, единственный способ узнать, что действительно нужно сделать — начать это делать. Это не значит начать делать что угодно. Стоит взять в работу основную часть, от которой зависят все остальные, причём такого размера, чтобы получить первый работающий результат за несколько дней.

В следующих главах мы узнаем, как команды выбирают, что брать, и как выстраивают совместную работу.

## Один кусок готов

---



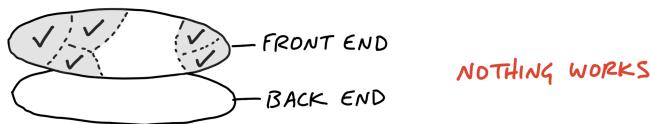
По мере того, как команда осваивает проект и начинает работу, она «обнаруживает» и учитывает задачи, которые нужно сделать, чтобы реализовать проект. Никто не создаёт окончательный план проекта, перечисляющий все его задачи. Если множество задач завершено, а потрогать руками нечего, нет ощущения движения вперёд. Несмотря на затраченные усилия, команда будет испытывать тревогу, поскольку не может показать результат. Много сделано, но ничего не готово.

Вместо этого мы просим команды стремиться сделать что-то готовое к показу как можно раньше — в течение примерно недели. Это значит глубоко проработать один кусок, а не начинать работу над несколькими кусками.

## Каждый кусок как проект

Многие проекты можно разбить на слои: фронт-энд и бэк-энд, или дизайн и код. Строго говоря, слоёв больше, но эти два слоя труднее всего интегрировать.

Допустим, в начале проекта команда много работает над дизайном, рисует экраны или даже пишет для них шаблоны. Но пока нет связи с бэк-эндом, ничего не работает по-настоящему.



Обратное также верно. Можно завершить много задач по бэк-энду, но пока нет интерфейса, работать с результатом нельзя. Как понять, что часть проекта, которая у тебя в работе, действительно решает свою задачу, если нельзя её потрогать?



Вместо этого, возьмите один кусок проекта и проработайте его на всех слоях одновременно. В результате у вас будет что-то, пусть небольшое, но работающее (или явно не работающее и требующее переделки). Это небольшое уже можно показать другим и попробовать самим — то ли это, что мы имели в виду.



## Пример: внешние наблюдатели в проектах Basecamp

Мы работали над фичей, которая бы позволяла пользователям Basecamp давать внешним наблюдателям (например, своим заказчикам) доступ к своим проектам — документами, сообщениям, спискам задач. Презентация проекта включала в себя:

- **Частичный доступ.** До этого, пользователь либо имел доступ ко всему, либо не имел его совсем. Нужна была возможность выдавать доступ к части проекта. Это, конечно, требовало больших изменений в бэк-енде, кешировании и прочем.
- **Управление наблюдателями.** Такими пользователями нужно было управлять отдельно от обычных участников проектов Basecamp.
- **Переключатель внешнего доступа.** У каждого кусочка проекта в Basecamp должен был появиться переключатель — доступен ли он для наблюдателей.

В команде был один дизайнер и один разработчик. Обдумав проект и изучив существующий код, дизайнер решил начать с переключателя. С точки зрения интерфейса это было

центральное изменение. Именно с ним чаще всего будут взаимодействовать пользователи.

Дизайнер не стал сразу рисовать чистовой дизайн, вместо этого он начал экспериментировать с HTML-шаблонами продукта. Что лучше, две радиокнопки, чекбокс, кнопка?

Тем временем, разработчик не сидел на месте. В презентации было достаточно информации для начала его работы по предоставлению доступа.

Как только дизайнер примерно определился с механикой переключателя, он показал его разработчику. Разработчик отложил в сторону работу по предоставлению доступа и добавил «сырой» переключатель во все требуемые страницы продукта, реализовал изменение состояния при клике и сохранение состояния в базе.

В этот момент нажатие на переключатель не меняло доступ. Но с точки зрения пользователя в моменте всё работало. Можно было пожить с этим решением, проверить его на реальных данных (на тестовом сервере, разумеется).

У дизайнера ещё было что поправить, но привлекать разработчика уже не требовалось. Дизайнер продолжил прорабатывать расположение, размер, мобильную версию и всё остальное, а разработчик вернулся к предоставлению доступа.

Через три дня после начала проекта, дизайнер продемонстрировал переключатель менеджеру. После демо было ещё несколько мелких правок, и переключатель отметили как «готовый».

Важный кусок проекта прошёл через дизайн, разработку, демо и правки. За три дня команда смогла добиться результата, который можно потрогать — это приятно, к тому же добавило уверенности, что проектное решение действительно работает.

Этот пример показывает, как команда строит работу вокруг одного и только одного куска проекта.

## **Разработчикам не приходится ждать**

Поскольку основные вопросы раскрыты в презентации, разработчикам не приходится ждать готового дизайна.

Информации достаточно, чтобы сразу начать работу над бэкендом.

## **Сначала просто элементы интерфейса, потом чистовой дизайн**

Разработчикам не нужен чистовой дизайн, чтобы начать реализацию. Достаточно примитивных элементов — полей ввода, кнопок, блоков с данными.

Дизайнер может работать с цветами, шрифтами и размерами позже, когда основные элементы интерфейса уже на месте и интегрированы с серверным кодом. Немного текста, интерактива и интеграция с сервером — это всё, что нужно, чтобы потрогать работающее решение в браузере.

Тогда ответы на самые важные вопросы — это вообще то, что мы хотели? Легко ли в этом разобраться? Делает ли это то, что нужно? — можно получить, не дожидаясь чистового дизайна.

Вот пример первой итерации интерфейса, которую дизайнер передаёт разработчику. Выглядит, как будто стили не загрузились :-)

The screenshot shows a web browser window with a light gray header bar. Below it is a white form area. At the top of the form, the text "Please choose an option" is displayed in bold. There are two radio buttons: one checked (blue outline) labeled "Private Room" and one uncheckable (gray outline) labeled "Dorm Room". Below this section, the heading "Arrival and departure" is shown in bold. Under "When do you arrive?", there is a date input field containing "Friday May 24, 2019" with a small calendar icon to its right. To the right of the date field is the text "in the" followed by a dropdown menu set to "Afternoon", with a small arrow icon indicating it's a dropdown. Under "When do you leave?", there is another date input field containing "Monday May 27, 2019" with a similar calendar icon. Below these sections, the heading "Your information" is shown in bold. There are two text input fields: one for "First name" and one for "Last name", both represented by simple rectangular boxes.

Это скриншот страницы регистрации на курсы, занимающие несколько дней. Дизайнер написал HTML вручную. Стилей почти нет — только простая визуальная иерархия.

В этой примитивной странице, тем не менее, видны важные интерфейсные решения:

- просим указать время прибытия, но не время выезда — это результат долгих предварительных обсуждений;
- набор вариантов в списке времени прибытия соответствует договорённостям о питании и позднем выезде из отеля;

В первом наброске дизайнер использовал календарную сетку для выбора дат. Но с ней возникло много проблем — например, некоторые курсы занимают несколько недель и включают

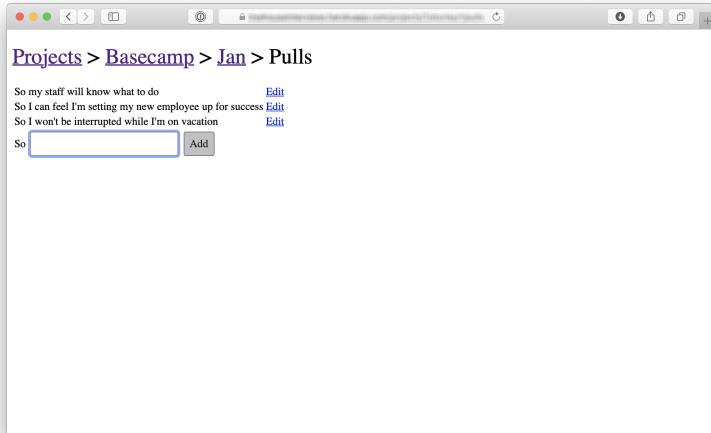
несколько этапов. Обычный выпадающий список оказался более подходящим в этой ситуации.

Вот ещё один пример — первый работающий кусок программы для ведения заметок во время интервью с пользователями.

The screenshot shows a web-based application window titled "Projects > Basecamp > Jan". At the top, there are input fields for "Name: Jan", "Story #: --", "A: --", and "B: --", followed by a link "Edit story definition". Below these are six columns labeled "Setup", "Pushes", "Pulls", "Habits", "Anxieties", and "Observations". The "Pushes" column contains a bulleted list: "• When I just hired another person", "• When I'm going to be on vacation", and "• When my staff has been losing track of information". The "Pulls" column contains a bulleted list: "• So my staff will know what to do", "• So I can feel I'm setting my new employee up for success", and "• So I won't be interrupted while I'm on vacation". Below each column are edit links: "Edit Setup", "Add/edit Pushes", "Add/edit Pulls", "Add/edit Habits", "Add/edit Anxieties", and "Edit Observations".

На этом этапе название проекта и тема интервью заполнены вручную и ни одна ссылка не работает. Дизайна как будто и нет — стандартные ссылки синего и фиолетового цветов, чёрные рамки колонок. Тем не менее, страница позволяет проверить важные компромиссы.

Дизайнер принял решение показать секции колонками, чтобы можно было увидеть как можно больше данных без необходимости скроллить. По этой причине на экране не осталось места для того, чтобы добавлять, редактировать и удалять данные в каждой секции. И поэтому дизайнер решил вынести эти действия в отдельные страницы.



Вот первый дизайн интерфейса для добавления и редактирования одного из типов данных. Опять же, это с трудом можно назвать «дизайном», но этого достаточно, чтобы интегрировать страницу с бэк-эндом и получить работающий кусок проекта. Команда может попробовать эти страницы в деле и понять, верным ли было решение выносить добавление данных в отдельную страницу. Если окажется, что нет — не было потрачено впустую время на чистовой дизайн.

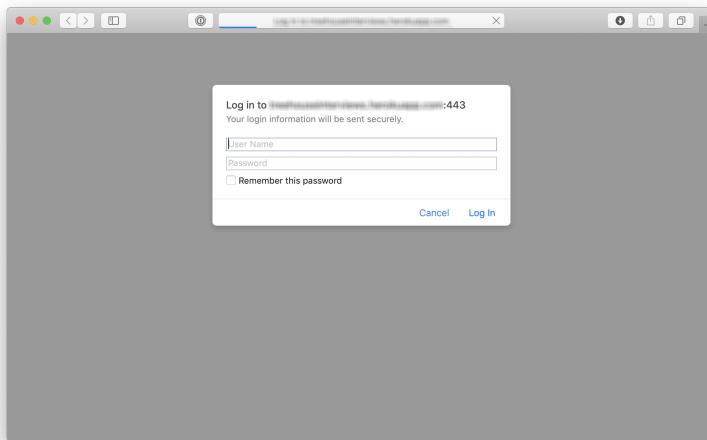
Цвета, шрифты и размеры не имеют значения на первом этапе работы. Прежде всего нужно понять, работает ли задумка в принципе. После того, как фронт-энд и бэк-энд интегрированы, дизайнер продолжает улучшать то, что уже работает.

**Программируйте только то, что нужно для следующего шага**

Тот же принцип работает и для бэк-енда. Не стоит пытаться сделать всё и сразу. Иногда дизайнеру для начала работы достаточно прикрутить к базе пару простых форм, или отдать в результатах запроса реальные данные.

Как и с фронт-ендом, результат первого подхода к задаче по бэк-енду может быть сырым, и это нормально. Например, контроллер уже формирует шаблоны, но модели ещё нет.

Ещё пример — упомянутая выше программа (заметки во время интервью с пользователями). Когда пришло время обсуждать первые результаты, команда знала, что будет работать с конфиденциальными данными. Нужна была какая-то защита. Вместо того, чтобы делать свою (или внедрять стороннюю) полноценную систему с логинами и паролями, разработчик ограничился простейшим HTTPAuth с зашитым в код паролем.



Сэкономленное время команда потратила на проверку интерфейса на реальных данных. Нормальная система авторизации выглядела бы лучше, но никак не приблизила бы проект к завершению.

В общем, идея в том, чтобы команда плотно работала над одним и только одним куском, не дожидаясь друг от друга закрытых задач, а постоянно обмениваясь мнениями и сырыми материалами. Скорее создайте что-то, что сможете вместе потрогать и понять, то ли это, что нужно.

## Начните с середины

В примере выше интересно то, что команда не начала работу над проектом со страницы входа или добавления интервью в базу. Они начали с середины, и уже потом занялись всем вокруг.

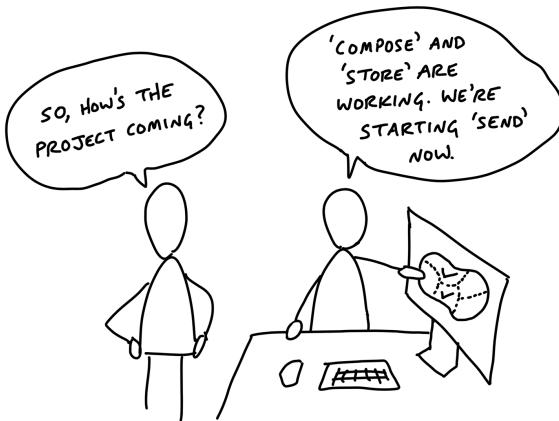
Вот три критерия, чтобы решить, с чего начать:

1. Задача должна содержать суть проекта. Переключатель доступа был сутью проекта Внешних наблюдателей, без него остальная работа не имела смысла. Задача «создать форму переименования наблюдателя» тоже была обязательна, но не в ней было главное. В проекте заметок процесс добавления заметок — определяющий, хоть и находится где-то в середине сценария использования.
2. Задача не должна быть слишком большой. Важно, чтобы первая задача была завершена достаточно быстро, это придаст команде уверенность и ускорение.

3. Задача должна быть нетипичной. Если несколько задач подходят под критерии выше, выбирайте то, с чем раньше не сталкивались. В проекте Внешних наблюдателей была задача «интерфейс добавления клиентов» — подобное команда делала десятки раз. Взять её в работу первой означало бы отложить задачи с большей неопределённостью. Вместо этого, взяв и завершив нетипичную задачу с переключателем доступа, команда уменьшила риск и доказала, что новая идея работает как было задумано.

# Карта проекта

---



В прошлой главе мы начали работу над проектом, выделив один осмысленный кусок и проработав его до полной готовности. Этот приём — часть более общей техники организации работы над проектом.

## Организуйте работу исходя из структуры проекта, а не ролей

Когда организуют работу по проекту, часто группируют задачи по исполнителям или ролям. Например, ведут список задач для дизайнеров и для разработчиков. Это приводит к проблеме, которую мы затронули ранее — люди завершают задачи, но задачи не складываются в единое решение.

Представьте, что вы организуете благотворительный вечер. Вы можете создать отдельный список дел для каждого из трёх

волонтёров, но тогда вы не сможете быстро увидеть общую картину и понять, как движется проект в целом — что готово, а что нет.

Вместо этого вы можете сгруппировать дела в соответствии с тем, из каких частей состоит проект, его структурой. Тогда частями можно управлять более независимо. Например, вы создадите отдельные списки дел для питания, освещения и звука, подготовки сцены, и так далее. Тогда, взглянув на эти списки, вы быстро поймёте, какие части готовы, а какие нет.

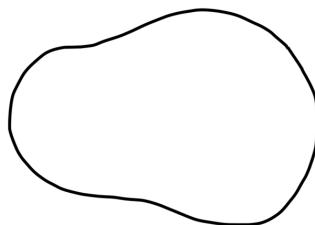
В разработке программных продуктов подобная структура часто неизвестна заранее. Мы создаём вещи, которых раньше не делали. Каждый проект — неизведанная территория, которую нужно обойти, чтобы нарисовать карту. Только начав работу, мы понимаем, что с чем связано, а что можно делать независимо.

В прошлой главе мы отметили, как независимые куски проекта включают себя интегрированные слои (обычно фронт-енд и бэк-енд). Только полностью закончив один кусок, мы переходим к следующему. Это намного лучше, чем работать над разными кусками и надеяться, что к концу цикла всё волшебным образом соединится в работающую систему.

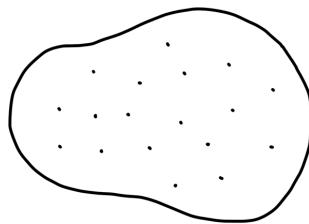
Давайте называть такой независимый кусок проекта «участком», по аналогии с участком земли. Мы берём целый участок проекта и нарезаем его на отдельные участки, которые могут быть реализованы независимо. В этой главе мы рассмотрим, как команда создаёт карту участков и работает с ними один за другим.

## Карта участков

Представьте себе проект с высоты птичьего полёта. Вначале обозначены только границы проекта целиком. Они — результат предыдущей работы по формированию проекта. Внутри этого участка нет никаких задач или разделителей.

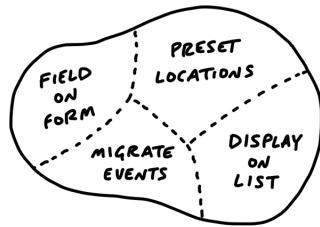


Начав работу, команда обнаруживает и создаёт задачи. Вполне естественно начинать с задач, они конкретны и неделимы. Пока ещё рано их группировать. В этот момент наша карта — это границы участка всего проекта, на котором там и сям натыканы точки-задачи.



Но это ещё не конец. С высоты не видна общая картина, задача — слишком мелкая деталь на карте. По мере работы (но не сразу!) команда понимает, как разные

задачи связаны друг с другом. Начинает вырисовываться структура проекта, задачи можно сгруппировать, обозначив границы отдельных участков.



Участки обозначают осмыслиенные куски проекта, над которыми можно работать независимо. Важно, что они небольшие — несколько дней работы. Намного меньше, чем целый проект.

Карта — это визуальный образ. На практике мы определяем участки при помощи отдельных списков задач. Каждый участок — это название списка. Внутри списка — задачи этого участка.

The screenshot shows a web browser window for 3.basecamp.com. At the top, there are navigation links: Home, Pings, Hey!, Activity, My Stuff, Find, and a user profile icon. Below the header, a banner says "Replace Geocode with Location Summary". The main content area is titled "To-dos" with a count of 724. A green button labeled "+ New list" is visible. There are four sections of tasks:

- Field on Form**: 0/3 completed.
  - Design a location\_summary field below the existing location textarea
  - Add hint text that shows an example of what a good summary looks like
  - Make the summary a required field
- Preset Locations**: 0/3 completed.
  - Stop copying geocode fields
  - Copy location\_summary from center to event on create
  - Make sure location\_summary copies when updating an existing event
- Display on List**: 0/3 completed.
  - Determine if partial list uses same code as widget list
  - Send location\_summary as venue in the widget JS
  - Design a state for truncated summaries if they don't fit in the table column
- Migrate Events**: 0/2 completed.
  - Populate location\_summary field based on city and state geocode
  - Add new column on centers and events

At the bottom right of the list area is a blue circular icon with a question mark.

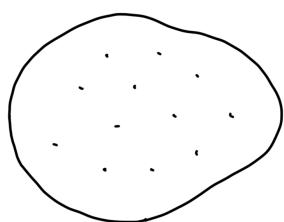
«Участки» — это не просто куски проекта. Их названия становятся терминами, при помощи которых команда общается. Например, «когда Частичный доступ будет готов, можно заняться Приглашением Наблюдателей. А обновим Сохранение видимости после того, как подключат Переключатель доступа».

Отчитываясь о результатах, команда использует эти термины, чтобы рассказать, что в каком статусе. Оперировать

подобными названиями участков на обсуждении верхнего уровня намного удобнее, чем опускаться на уровень отдельных задач. (Подробнее об отчётах — в главе, посвящённой графикам-холмам).

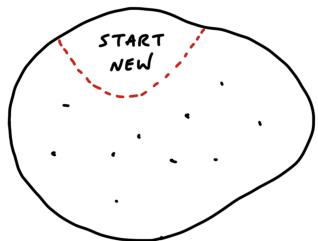
## Пример проекта: Черновики сообщений

Команда из дизайнера и разработчика работала над проектом создания и сохранения черновиков сообщений. После презентации они встретились и обозначили некоторые задачи, которые сразу пришли им в голову. Список дел назвали «Unscoped» — участков ещё нет.



- ≡  **Unscoped**
- ≡  Intercept attempts to reply to Topic if a draft from a different message exists
- ≡  Handle draft message timestamps after sending
- ≡  Hook up Send from Draft edit state
- ≡  Remember draft content when editing draft
- ≡  Remember addresses when editing draft
- ≡  Hook up re-saving from Draft edit state
- ≡  Reduce duplication in Draft forms
- ≡  Design index of existing Drafts
- ≡  Hook up Draft deletion from Draft edit state
- ≡  Design "Only you can see this draft" wrapper/labels
- ✓  Allow 'invalid' drafts to be created (without an addressee, for example)
- ✓  Hook up manual Draft creation

К концу первой недели работы они завершили некоторые задачи, но показывать было нечего. Следуя правилу «заверши один кусок», они обозначили один участок — «Создание черновика» (Start new) — и сфокусировались на нём.

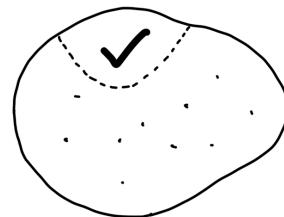


- ≡ ● **Start New**
- =  Design "Only you can see this draft" wrapper/labels
- =  Allow "invalid" drafts to be created (without an addressee, for example)
- =  Hook up manual Draft creation

- ≡ ● **Unscoped**

- =  Intercept attempts to reply to Topic if a draft from a different message exists
- =  Handle draft message timestamps after sending
- =  Hook up Send from Draft edit state
- =  Remember draft content when editing draft
- =  Remember addresses when editing draft
- =  Hook up re-saving from Draft edit state
- =  Reduce duplication in Draft forms
- =  Design index of existing Drafts
- =  Hook up Draft deletion from Draft edit state

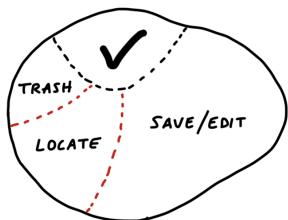
Некоторые задачи из «Unscoped» переместились в список «Создание черновика». Оказалось, что для завершения этого участка осталось закончить только одну дизайн-задачу. После этого участок был готов.



- ≡ ● **Unscoped**

- =  Intercept attempts to reply to Topic if a draft from a different message exists
- =  Handle draft message timestamps after sending
- =  Hook up Send from Draft edit state
- =  Remember draft content when editing draft
- =  Remember addresses when editing draft
- =  Hook up re-saving from Draft edit state
- =  Reduce duplication in Draft forms
- =  Design index of existing Drafts
- =  Hook up Draft deletion from Draft edit state

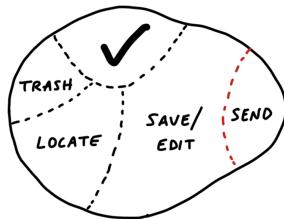
Глядя на оставшиеся в списке задачи, команда решила выделить ещё несколько участков — задачи, посвящённые поиску, ушли в участок «Поиск» (Locate), задачи про удаление — в участок «Удаление» (Trash). Всё, что осталось, пока что примерно назвали «Сохранить или редактировать» (Save/Edit).



- ≡ ● **Save/Edit**
- ≡  Intercept attempts to reply to Topic if a draft from a different message exists
- ≡  Handle draft message timestamps after sending
- ≡  Hook up Send from Draft edit state
- ≡  Remember draft content when editing draft
- ≡  Remember addresses when editing draft
- ≡  Hook up re-saving from Draft edit state
- ≡  Reduce duplication in Draft forms
  
- ≡ ● **Trash**
- ≡  Hook up Draft deletion from Draft edit state
  
- ≡ ● **Locate**
- ≡  Design index of existing Drafts

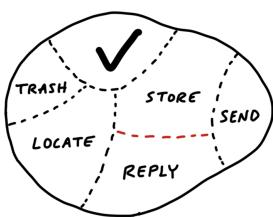
Обратите внимание на участок «Поиск» (Locate). Сейчас там только одна задача (список черновиков). Но когда дойдёт до дела, там точно появятся новые задачи.

Итак, дизайнер взял в работу участок «Поиск», а разработчик — «Сохранить или редактировать». По мере работы она обнаружила, что можно поделить участок на два, разных по смыслу, чтобы быстрее достигнуть видимого результата по каждому из них. Задачи, относящиеся в отправке, она перенесла в новый участок — «Отправка» (Send).



- ≡ **Send**
  - Hook up Send from Draft edit state
  - Handle draft message timestamps after sending
  
- ≡ **Save/Edit**
  - Intercept attempts to reply to Topic if a draft from a different message exists
  - Remember draft content when editing draft
  - Remember addresses when editing draft
  - Hook up re-saving from Draft edit state
  - Reduce duplication in Draft forms
  
- ≡ **Trash**
  - Hook up Draft deletion from Draft edit state
  - Design a way to trash drafts from the index of drafts
  
- ≡ **Locate**
  - Design index of existing Drafts
  - Design a way to navigate to Drafts via "Inbox..." menu

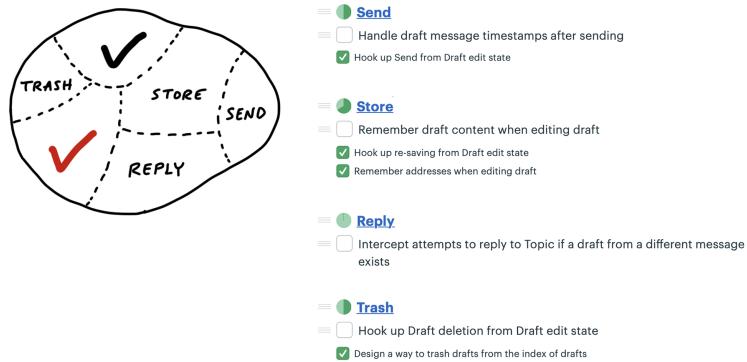
Наконец, пара оставшихся задач в «Сохранить или редактировать» была на самом деле про хранение информации, и ещё одна задача была про обработку особой ситуации при ответе на другое сообщение. Разработчик переделала участки таким образом:



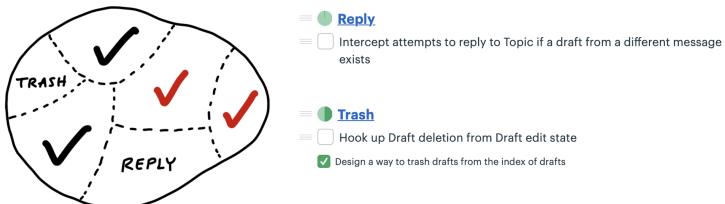
- ≡ **Send**
  - Hook up Send from Draft edit state
  - Handle draft message timestamps after sending
  
- ≡ **Store**
  - Remember draft content when editing draft
  - Remember addresses when editing draft
  - Hook up re-saving from Draft edit state
  
- ≡ **Reply**
  - Intercept attempts to reply to Topic if a draft from a different message exists
  
- ≡ **Trash**
  - Hook up Draft deletion from Draft edit state
  - Design a way to trash drafts from the index of drafts
  
- ≡ **Locate**
  - Design index of existing Drafts
  - Design a way to navigate to Drafts via "Inbox..." menu

И в этот момент у команды возникло понимание — что именно представляет из себя проект на верхнем уровне. Все части по отдельности имели смысл и были логично связаны друг с другом. Ни одна часть не была слишком большой, чтобы потерять прогресс или не заметить что-то важное.

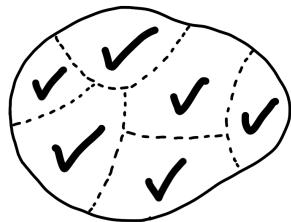
Тем временем дизайнер работал над участком «Поиска». После небольшой интеграции, этот участок отметили как готовый. Задачи завершались также в участках «Отправки» и «Хранения».



Когда эти участки были завершены, осталась лишь пара задачи в участках «Корзина» и «Ответ».



И вот, весь проект был завершён.



## Обозначение участков

Обозначение участков — не «планирование проекта». Надо пройти по территории, прежде чем рисовать её карту. Должным образом обозначенные участки — не просто группы задач или категории, призванные разбить слишком длинный список. Они отражают смысл частей, над которыми можно работать независимо, и связей между этими частями.

Выделив связанные задачи в участок, можно сказать, когда весь участок «готов». Но эти связи часто не видны заранее. Помните про разницу придуманных и обнаруженных задач? Тот же принцип применим и к участкам. Границы участков вырисовываются в процессе работы и наблюдения за тем, как связаны части проекта.

Вот почему в начале проекта никто не ожидает чётко распланированных участков. Чаще всего они готовы к концу первой или началу второй недели, после того как команда немного поработала и прошла по территории.

Также нормально переделывать участки в процессе. Их размер и названия меняются по мере того, как команда всё лучше

понимает проект. В примере выше группа задач, посвящённая отправке черновиков, не была видна сразу, и только когда разработчик начала работать на участке Сохранения и редактирования, она обнаружила независимость этих задач и выделила для них отдельный участок.

## Как понять, верно ли размечены участки

Хорошо размеченные участки соответствуют анатомии проекта. Вы легко можете объяснить другому человеку, где у вас болит, поскольку у частей тела есть общепринятые названия (руки, ноги, голова), которые соответствуют смысловым частям, анатомии. У каждого проекта есть подобная анатомия, продиктованная его архитектурой, окружением и взаимосвязями.

Есть три признака того, что участки размечены хорошо:

1. Вы чувствуете, что видите состояние проекта целиком, и ничто важное не скрыто на уровне отдельных задач.
2. Названия участков помогают в обсуждении проекта, делают их легче и понятнее, а не наоборот.
3. Вы сразу понимаете, в какой участок добавить каждую новую задачу.

И наоборот, вот три признака того, что участки стоит переразметить:

1. Трудно понять, насколько готов участок. Это часто происходит, когда задачи внутри участка не взаимосвязаны. Тогда завершение одной задачи никак

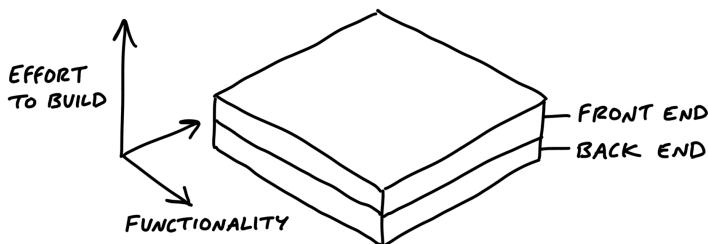
не влияет на готовность других. В этом случае стоит выделить некоторые задачи в отдельные участки.

2. Название участка слишком общее, например «фронт-енд» или «баги». Это знак того, что участок недостаточно независим или не отражает какую-то смысловую часть проекта. Например, вместо участка «Баги» мы распределяем их по участкам исходя из содержания — этот баг Отправки, этот Хранения, и так далее.
  3. Участок слишком велик — он становится похож на целый проект с одним длиннющим списком задач. Лучше разбить такой участок на несколько, каждый из которых можно быстро довести до готовности.
- 

Напоследок — пара советов о разных видах задач и участков.

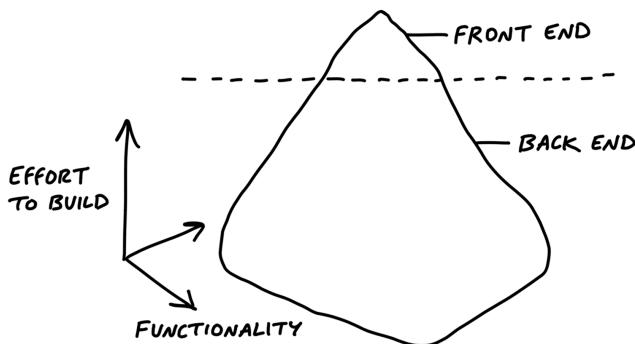
## Торты

Чаще всего проекты требуют два «слоя» работы — дизайн (фронт-енд) и серверный код (бэк-енд). Например, часть приложения принимает данные от пользователя, сохраняет их в базе и отображает в каком-то виде. Работа над такими приложениями выглядит как торт из двух примерно одинаковых слоёв — объем работы над бэк-ендом можно оценить, глядя на дизайн. Поэтому можно смело интегрировать задачи обоих слоёв в один участок.



## Айсберги

Но иногда ресурсы а бэк-енда требуется намного больше фронт-енда (или наоборот). Например, дизайн состоит только из одной небольшой формы, но обработка этой формы требует сложной бизнес-логики. Такая работа скорее напоминает айсберг.



В таких случаях мы выделяем фронт-енд в отдельный независимый участок (при условии, что интерфейсные решения не зависят от серверных). А если бэк-енд слишком сложен для одного участка, стоит разбить его на части.

Основная цель этих упражнений — выделить вещи, которые можно быстро довести до готовности, и не доводить проект до ситуации, когда работа по многим участкам должна завершиться в последний момент.

Изредка бывает айсберг «наоборот» — сложный фронт-энд и простой бэк-энд. Например, в модели данных календаря нет ничего особого, но зато есть куча фронт-енда, например, обработка событий длиной в несколько дней, привязка к сетке, и так далее.

Натыкаясь на любые айсберги, мы подвергаем их сомнению, прежде чем начать работу. А точно нужна такая сложность? Точно не осталось способов её уменьшить? Насколько необходим такой богатый интерфейс? Можно ли выстроить серверные процессы с меньшим количеством зависимостей?

## **Винегрет**

Почти всегда находится пара задач, которую не засунешь ни в один участок. Мы позволяем себе добавить участок под названием «Винегрет» для таких задач. Но мы пристально следим на его содержанием — если в нём больше 3-5 задач, мы что-то сделали не так и какой-то участок ждёт, пока его заметят.

## **Отмечайте необязательные задачи**

По мере изучения проблемы вы всё время будете создавать новые задачи. Какой-то код лучше бы почистить, какой-то нестандартный случай — обработать. Мы отмечаем задачи,

которые «лучше бы» сделать, префиксом «~». Так команда сразу видит, какие задачи обязательны, а какие нет.

В мире с бесконечным бюджетом и сроками, можно улучшать всё без перерыва. Но в реальности, нам нужно постоянно обрубать щупальца растущего объёма работ. Знак «~» в начале названия задачи или целого участка — наш способ сделать это. (Подробнее в главе 14).

# Демонстрируйте прогресс

---



Хороший менеджер не любит спрашивать, как продвигается работа. Неловко отвлекать тех, кто занят работой, и особенно неприятно, когда полученный ответ требует дополнительных вопросов.

Удобнее, когда есть место, которое в любой момент показывает актуальный статус. В прошлой главе мы узнали, как группировка задач в «участки» помогает команде понимать, что они делают. Но менеджеру эта информация не так полезна. Есть причины, по которым использовать списки задач для понимания статуса неэффективно.

## Задачи ещё не сформулированы

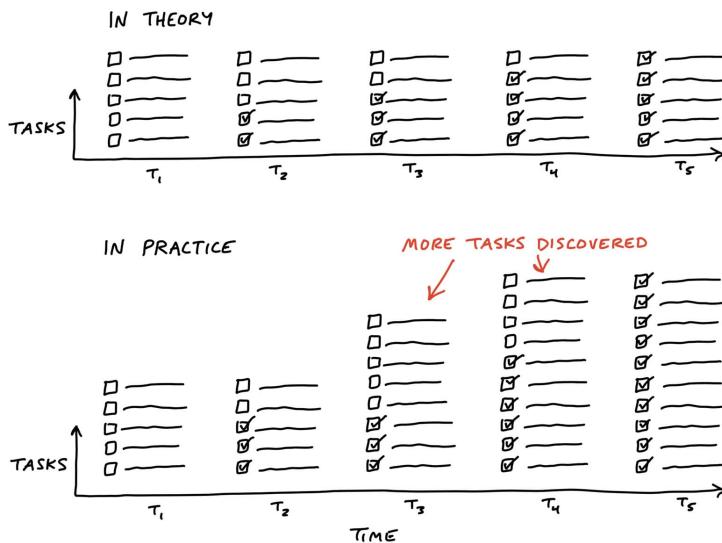
Допустим, менеджер видит список, в котором несколько завершённых задач и ни одной незавершённой. Как будто вся работа сделана. Однако может быть, команда знает, что ещё нужно сделать, просто пока не сформулировала задачи.

Иногда команда обозначает участок раньше, чем заполняет его задачами. Так она формулирует, «что» нужно сделать, пока не углубляясь в то, «как».

Ещё пример — тестирование в конце цикла разработки.

Тестировщики добавляют задачи в участки, которые до этого были полностью «готовы».

Вспомните о разнице «придуманных» и «обнаруженных» задач. Наивно полагать, что можно заранее составить список дел, делать их и ставить галочки — одну за другой. В реальности, списки задач растут по мере работы над проектом.



В примере выше, если бы менеджер хотел узнать, как идут дела, в момент  $T_2$ , список задач ввёл бы его в заблуждение. Сторонний наблюдатель ничего не знает о том, больше или меньше станет невыполненных задач. Чтобы предсказать это,

нужно быть в контексте — понимать, что конкретно готово и какие новые задачи могут возникнуть.

## **Оценки не учитывают неопределённость**

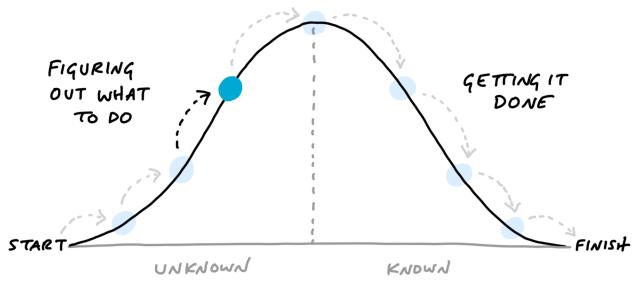
Некоторые команды добавляют числовую оценку к каждой задаче или участку. Проблема с оценкой — в том, что у разной по сути работы оценка будет означать совершенно разные вещи.

Допустим, есть 2 задачи, каждую из которых оценили в 4 часа. При этом первая задача похожа на то, что команда уже делала много раз, значит, оценке можно доверять. А вторая задача — совершенно новая. Если всё пойдёт по плану, действительно заняло бы 4 часа, но на практике вышло 3 дня. А оценка «от 4 часов до 3 дней» не имеет смысла.

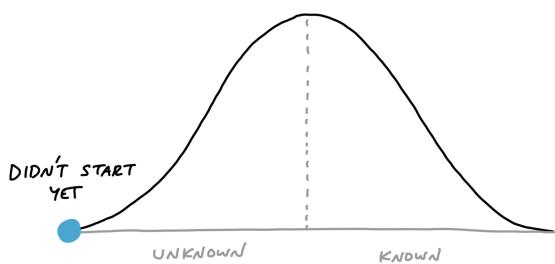
Мы придумали свой способ показывать статус работы, без статистики задач и без числовых оценок. Вместо разделения «не сделано / сделано» мы обращаем внимание на «неясно / ясно». Для работы с этой метрикой мы используем метафору холма.

## **Работу сделать — холм преодолеть**

У каждой работы есть две фазы. Первая — движение вверх по холму, выяснение, что именно нужно сделать, и как конкретно это сделать. На вершине холма — полное понимание предстоящих действий. И вторая фаза — движение вниз по холму, выполнение действий.



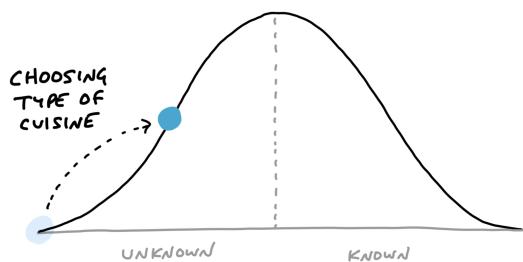
Вот пример. Вы планируете организовать вечеринку. Вы уже знаете дату, до неё ещё пара недель и вы не начали продумывать, что будете готовить. Вы — в начале пути.



Вы составляете список гостей, и обращаете внимание, что пара из них — вегетарианцы. Гриль придётся вычеркнуть, но есть много других вариантов. Вы размышляете про итальянскую или индийскую кухню, и склоняетесь к индийской, она вам интереснее.

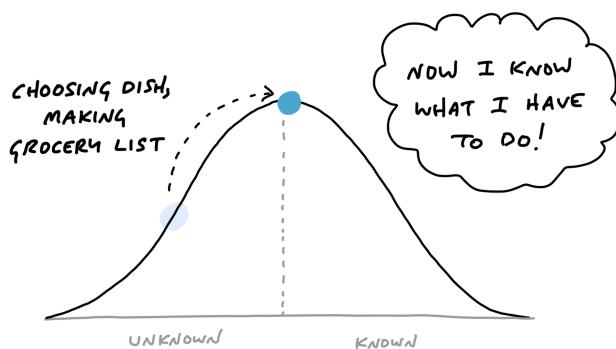
В этот момент вопрос «на сколько процентов завершён проект?» не имеет смысла. Если кто-то попросит вас оценить время на шопинг и другую подготовку, вы не сможете — вы ещё не выбрали, что именно будете готовить. Правильный ответ такой — «я провёл работу по выбору типа кухни, но ещё не определился с точным набором блюд».

На нашем графике этому состоянию соответствует середина пути вверх по холму.



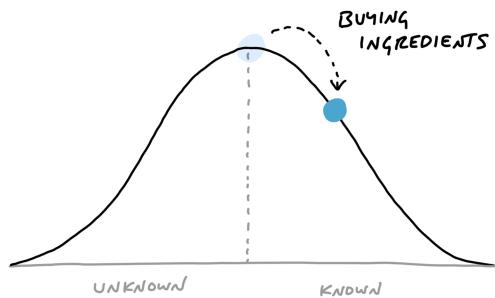
Вы изучаете рецепты и выбираете блюда, которые вам интересно приготовить, но чтобы не было слишком экзотических ингредиентов. В процессе поиска вы составляете список продуктов, которые нужно купить.

Это совсем другая ситуация. От «я ещё не знаю, что делаю» вы пришли к «теперь всё ясно, осталось пойти и сделать». Вы — на вершине холма.

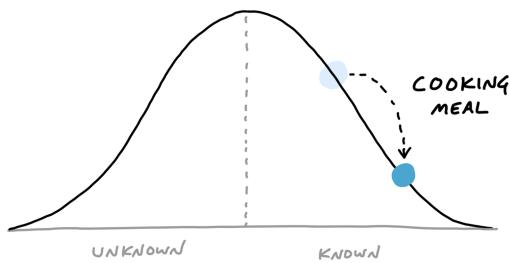


С этой точки вам видны все действия, которые осталось совершить. Теперь можно оценить время — пару часов на магазин, полчаса на разгрузку, час на готовку, и так далее.

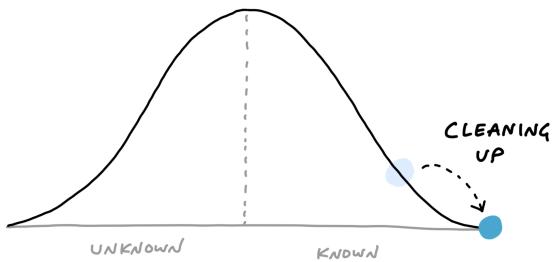
Накануне вечеринки вы едете в магазин и покупаете продукты. Вы — на пути вниз по холму.



Дальше, вы готовите еду и встречаете гостей.



Когда гости ушли, остаётся вымыть посуду.

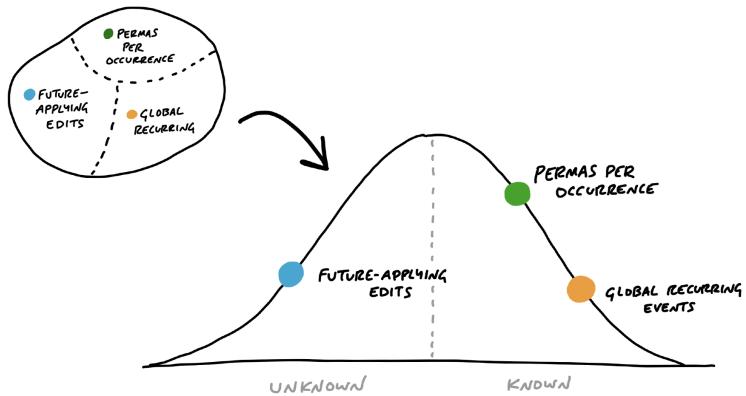


Обратите внимание — график-холм помогает показать разницу в ощущениях на разных стадиях работы. Путь вверх полон неопределённости, поисков ответов, решения проблем. Путь вниз — уверенность, ясность, полный контроль и понимание.

## Участки на холме

Мы можем объединить метафору «холма» с метафорой «участков» из прошлой главы. Разметив участки и дав им названия, мы говорим о частях проекта друг с другом одними и теми же словами. А перемещая участки по холму, мы сообщаем их статус (вверх или вниз).

Например, обозначим участки точками разных цветов на графике-холме:



Это — скриншот реального проекта по поддержке повторяющихся событий в Basecamp. Участок «Future-applying edits» всё ещё «карабкается наверх», там много неясного. С остальными двумя всё ясно, а «Global recurring events» уже близок к готовности.

## Отчёт о статусе без вопросов и ответов

Мы в Basecamp отслеживаем статус всех работ при помощи графиков-холмов. Участники команд, у которых есть полная информация по состоянию проекта, просто перетаскивают участки на подходящее место на холме.

[+ New list](#)

## To-dos

[View as... ▾](#)[Update](#)

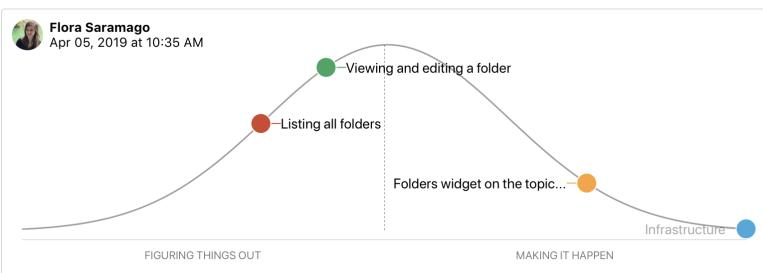
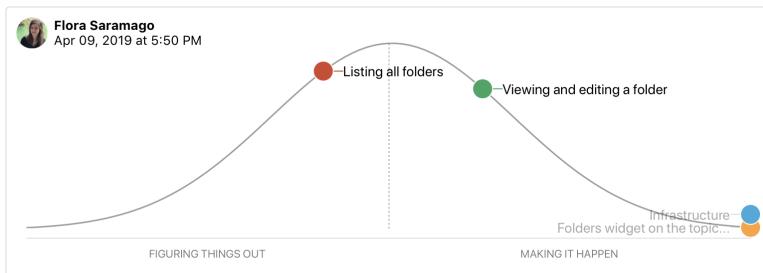
Last updated 7 minutes ago

-  Future-applying edits
-  Permas per occurrence
-  Global recurring events

FIGURING THINGS OUT

MAKING IT HAPPEN

У менеджеров появляется ценная возможность — видеть работу в динамике, не только где на холме работа прямо сейчас, но и как она движется во времени.



Сразу видно, есть ли где-то затык. Можно понять, над чем команда работает сейчас и сколько времени у них уходит на такую часть работы.

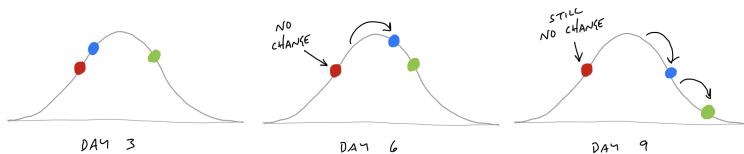
Этот график – основной источник информации по состоянию проекта для менеджера. График не требует от менеджера отвлекать команду вопросами. А если на графике видна проблема – менеджер может сразу начать предметный

разговор. «Похоже, что Autosave застряло на пути к вершине холма. В чём там неясность?».

## Не приходится говорить «Не знаю»

Никто не любит говорить «Я не знаю, как это сделать» (пусть даже пока). Упустить момент, когда работа встала или команда ходит кругами — самый большой риск. Этот момент важно поймать как можно раньше, тогда можно привлечь помочь или переосмыслить проблему. Иначе одна задача может поставить под угрозу весь проект.

График-холм деликатно показывает возможный затык, без необходимости исполнителю формулировать его. Ищите на графике точку, которая стоит на месте на пути вверх:

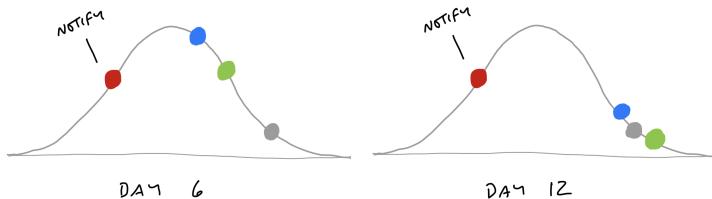


Терминология холма также помогает построить разговор более конструктивно — обсуждение строится вокруг задачи («как нам перевалить эту задачу через вершину?»), а не человека («похоже, ты застрял»).

## Слишком большие участки

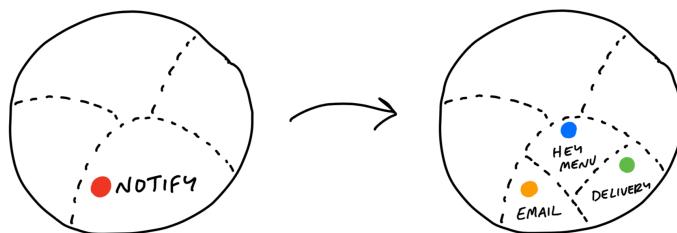
Иногда участок, застрявший по пути на вершину, просто слишком большой, и работа по нему идёт нормально, но не отражается в графике.

Однажды участок под названием «Notify» застрял на пути к вершине холма:



Пообщавшись с командой, менеджер узнал, что работа идёт хорошо, но фактически внутри участка спрятались три независимых куска — дизайн письма, отправка письма на бэкенд и показ уведомления. Первые два куска были готовы, а работа над уведомлением ещё не началась, поэтому участок целиком не менял статус.

Проблема решилась простым разделением одного участка на три.



== ● **Notify**

- Disable read receipts on the delivery service
- Use the batch delivery service
- Design icon for the Hey notification
- Figure out how to title it so it matches the Hey format
- Optimize the preview image for email
- Push the notification to the Hey menu
- Adjust email design so there's room for the document preview
- Write copy for the email

== ● **Hey Menu Notification**

- Figure out how to title it so it matches the Hey format
- Design icon for the Hey notification
- Push the notification to the Hey menu

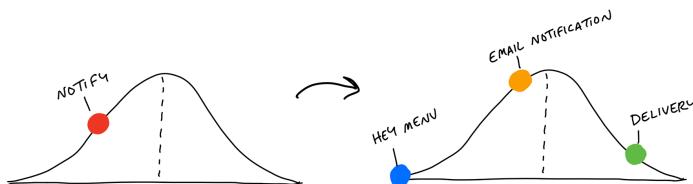
== ● **Email Notification**

- Optimize the preview image for email
- Adjust email design so there's room for the document preview
- Write copy for the email

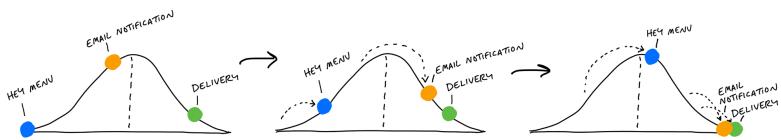
== ● **Notification Delivery**

- Disable read receipts on the delivery service
- Use the batch delivery service

Теперь каждый участок можно поместить на подходящее место на холме.



Теперь эти участки движутся по холму независимо, и менеджер может следить за изменением их состояния во времени.  
Профит!



## Проложите дорогу наверх

Когда люди начинают применять на практике графики-холмы, иногда им приходится возвращать точки на графике обратно, к началу или в левую половину. Это происходит, когда команда решает, что с работой всё ясно, а потом обнаруживает неразрешённый вопрос.

Частая причина этого — кто-то проделал подготовительную работу в голове, но не на практике. У нас часто есть иллюзия понимания («я просто использую вот это API»), но реальность оказывается более сложной.

Нам помогают вот такие подсказки — первая треть пути наверх означает «у меня есть мысли по этому поводу», вторая треть — «я проверил свои предположения», и последняя — «я сделал достаточно много, чтобы быть уверенным, что сюрпризов не будет».

## Задачи в правильном порядке

Кроме понимания состояния проекта, мы используем графики-холмы для того, чтобы брать «участки» в работу в правильном порядке.

Дело в том, что некоторые участки скрывают в себе больше риска, чем остальные. Представьте себе два участка:

геокодирование данных и рассылка уведомлений по почте. Оба участка содержат неопределённости, и оба участка сейчас находятся в начале пути по холму (работа не начата). Однако у команды совсем нет опыта в геокодировании. Спросите себя — если нам будет не хватать времени, какой участок безопаснее затащить на вершину первым?

Очевидно, первый, в котором больше риска. Команда в первую очередь передвигает первый участок наверх (разбирается с геокодированием, находит все решения), затем берётся за другие важные задачи, зная, что риск на участке Геокодирования минимизирован.

Любая работа стремится занять отпущененный ей объем времени. Если бы команда начала с уведомлений, можно было бы незаметно потратить недели, вычищая тексты и внося правки в дизайн. Можно, но не нужно. В последнюю неделю цикла вполне можно потратить день и сделать достаточно хороший шаблон письма. С другой стороны, если оставить на последний момент геокодирование, велик шанс, что в процессе работы возникнут проблемы, которые займут недели.

В журналистике есть термин «перевёрнутая пирамида». Смысл в том, что статья начинается с самой ценной информации в кратком виде, и уже потом идут подробности, раскрывающие тему. Это позволяет сверстать газетный разворот так, что самое важное можно прочитать сразу, не разворачивая газету.

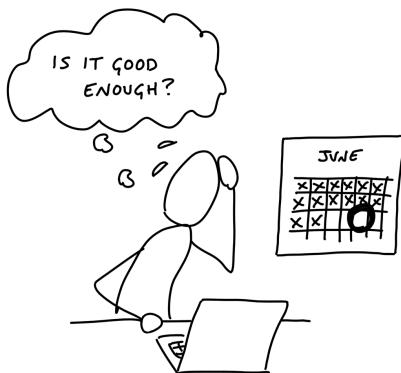
Разработчики используют подобный способ — «прогрессивный JPEG» — решая, в каком порядке брать задачи в работу. Сначала — дотащить самые важные задачи, с наибольшей

неопределённостью, до вершины холма. Спустить их с холма, проработав все детали, можно позже.

Приближается конец цикла, все важные задачи завершены, здесь и там остались кучки незавершённых задач с пометкой «~» (необязательно). В следующей главе мы обсудим, в какой момент стоит остановиться.

## Решите, когда остановиться

---



Следуя описанным выше приёмам, команда подходит к концу цикла в хорошей форме. Сформированный проект дал разработчикам чёткое направление, адекватно размеченные участки позволили разбить проект на удобные куски и довести их до готовности. Все важные задачи с наибольшей неопределённостью завершены, поскольку их взяли в работу в первую очередь.

Однако задач всегда больше, чем времени на них. Завершить проект всегда означает оставить что-то за бортом. Приходит время задать себе неприятный вопрос — может быть, сейчас уже достаточно хорошо? И то, что я сделал, готово к выходу в свет?

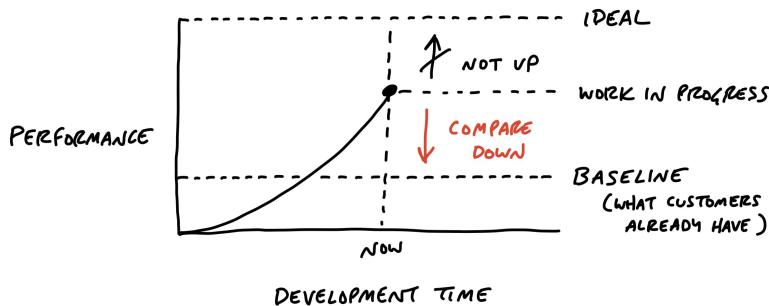
**Сравнивайте с текущей реальностью,  
не с идеалом**

Дизайнеры и разработчики хотят сделать всё наилучшим образом. Неважно, находится ли кнопка в самом центре лендинга или в конце длинной страницы настроек — дизайнер отнесётся к ней со всей тщательностью. Разработчики также стремятся к тому, чтобы код был максимально читаем, логичен и обрабатывал все возможные сценарии.

Подобная гордость важна для морального духа, но нужно направить её в верное русло. Стремясь к идеалу, мы не достигнем ничего. С другой стороны, не хочется понижать планку качества. Где та золотая середина, где можно уверенно сказать «теперь достаточно хорошо»?

Решение простое. Вместо абстрактного идеала сравнивайте результат своей работы с тем, как сейчас решают данную проблему пользователи, с их текущей реальностью. Какой костыль они сейчас используют? Насколько ваша работа улучшит их жизнь? Сколько им придётся терпеть, пока вы вычищаете все детали или сравниваете разные варианты дизайна?

Сравнение того, что мы успели сделать, с тем, как сейчас решают проблему пользователи, а не с идеалом, даёт нам настоящее ощущение прогресса. Становится легче отменять задачи, замедляющие проект. Меньше думаешь о себе, больше — о пользователе. Можно сказать себе: «Возможно, это не идеально, но точно работает как надо и для пользователей будет заметное улучшение».



## Ограничения диктуют компромиссы

Вспомним правило предохранителя — если проект не завершён за 6 недель, он отменяется.

Это ограничение заставляет команду идти на компромиссы. Каждый раз, задав вопрос «А не лучше ли...», мы сразу добавляем — а есть ли на это время? Жёсткий дедлайн не даёт пойти в работу задачам, которые не стоят затраченного времени.

Мы ожидаем, что команды будут активно срезать углы и подвергать все новые задачи сомнению, вместо того, чтобы работать по ночам в конце цикла.

## Списки дел растут сами по себе

Как сорняки. И это не вина заказчиков, менеджеров или разработчиков. Детали не видны заранее. Только начав работу, а иногда сделав заметную её часть, ты понимаешь, что ещё надо сделать.

Ещё, каждый проект полон надежд, которые не стоит воплощать. Каждой фиче необходимо быть самой эффективной, самой продуманной, самой проработанной. Не каждый сценарий использования одинаково важен для того рынка, которому вы надеетесь продать результат работы.

Такова жизнь. Не пытайтесь вручную остановить рост объёма работ. Вместо этого дайте командам инструменты и полномочия, чтобы они сами регулярно пропалывали сорняки.

## **Отменить задачу не значит ухудшить качество**

Решение не делать что-то, или делать частично, совсем не означает оставить дырку в продукте. Взвешенный выбор делает продукт лучше в целом, потому что он становится лучше *в чём-то* вместо чего-то другого. Набор принятых вами решений *отличает* продукт от конкурентов, создатели которых принимали другие решения.

Уменьшить объём работы не значит пожертвовать качеством. Мы чрезвычайно требовательны к качеству кода, дизайна, текста и производительности. Секрет в том, чтобы постоянно спрашивать себя — это точно имеет значение, точно ценно для пользователей?

## **Вколачивание объёма**

Часто говорят «урезать» объём работ. Мы используем более жёсткое слово — «вколотить» объём в сроки (scope

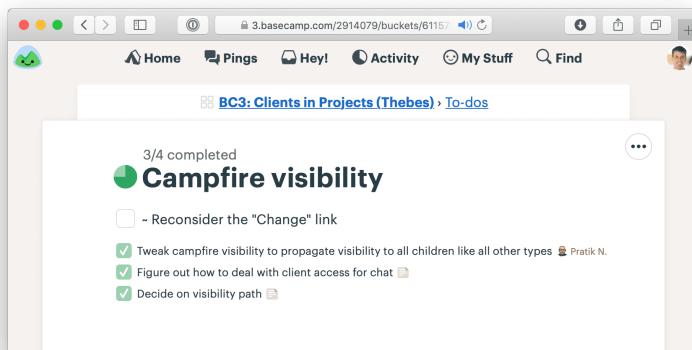
hammering) — для обозначения силы, которую надо применить к объёму работы, чтобы уместить его во временные рамки.

Вот вопросы, которые мы себе задаём:

- эта задача точно обязательна для результата?
- а без неё можно завершить проект?
- что будет, если задачу отменить?
- это что-то новое, или уже привычное для пользователей?
- как часто возникает этот сценарий или условие?
- это массовый сценарий, или нестандартный?
- если описанный случай возникает, каковы его реальные последствия?
- если что-то не так в описанном сценарии, насколько он вообще подходит для нашей аудитории?

Жёсткий дедлайн мотивирует нас задавать вопросы, а возможность менять объём работы — искать ответы и действовать. Вколачивая объём работы во временные рамки, ищем то, что действительно важно, и фокусируемся только на этом.

Для каждой новой задачи, команда решает, обязательная она или нет. Проект не завершён, пока не завершены все обязательные задачи. И наоборот, в завершённом проекте могут оставаться незавершённые необязательные задачи. Мы обозначаем их знаком «~» в начале названия. Если будет время — их возьмут в работу, если нет — их отменят. Обычно отменяют. Отметка «~» и есть вколачивание.



## Тестирование — только для нестандартных сценариев

В Basecamp сейчас работает один тестировщик (на дюжину человек в продуктовых командах и около миллиона пользователей). Он включается в работу ближе к концу каждого цикла и проверяет нестандартные сценарии, выходящие за рамки основного поведения.

Это возможно, поскольку дизайнеры и разработчики несут личную ответственность за качество своей работы.

Разработчики сами пишут тесты, и вся команда проекта участвует в проверке результата. Это — следствие того, что команда получила неограниченные полномочия и полную ответственность в момент передачи проекта (см. главу 9).

Много лет у нас вообще не было тестировщика. В какой-то момент количество пользователей выросло настолько, что некоторые сценарии, которые мы считали нестандартными, стали затрагивать сотни или тысячи пользователей.

Добавление процесса тестирования позволило улучшить обработку таких сценариев и снизить нагрузку на поддержку.

Поэтому мы считаем тестирование «переходом на новый уровень», а не этапом, который обязательно должен пройти каждый. Наличие тестировщика сильно нам помогает, но не является обязательным условием для создания качественного продукта.

По умолчанию тестировщик создаёт необязательные задачи (со знаком «~»). Команда отсматривает их и переводит в обязательные в зависимости от критичности и времени. Обычно тестировщик ведёт свой список (участок), а команда переносит взятые в работу задачи в участок, подходящий по смыслу. Так видно, какие участки снова требуют завершения.

Аналогично мы относимся и к проверке кода (code review). Команда имеет полномочия опубликовать код без ревью. Однако ревью делает код лучше, поэтому если есть время, один из руководителей может сделать ревью и дать обратную связь. Это скорее возможность обучения и прокачивания навыков, чем обязательный этап процесса.

## **А не продлить ли нам дедлайн**

Очень-очень редко мы позволяем проекту продлить срок сдачи на пару недель. В каких случаях?

Во-первых, незавершённые задачи должны реально быть «обязательными» и пережить все попытки вколовчивания объёма в сроки.

Во-вторых, вся оставшаяся работа должна находиться на правой половине графика-холма. Никаких неясных зависимостей, никаких открытых вопросов. Любая задача по пути к вершине холма в конце цикла — это следствие ошибки во время формирования проекта. В следующем цикле лучше поставить реализацию на паузу и вернуть проект автору на формирование.

Даже если оба эти условия выполнены, мы предпочитаем соблюдать дисциплину и не продлевать проекты. У команды есть возможность доделать все задачи во время двухнедельного перерыва между циклами — но это не должно становиться привычкой. Работа над задачами во время перерыва — симптом проблемы либо с проектом, либо с эффективностью команды.

# **Двигайтесь дальше**

---

## **Пусть ветер затихнет**

Релиз (публикация результатов) может создать кучу новой работы, если это допустить. Пользователи пишут: «супер, но мы просили не только это, а ещё то и то». Приходят сообщения об ошибках и предложения, что надо срочно улучшить. Команда хочет знать, как аудитория восприняла работу, и внимательно следит за обратной связью.

Особая жара — когда релиз меняет что-то в привычном поведении пользователей. Даже чисто визуальные изменения часто вызывают волны гнева. То и дело прилетают отзывы вида «Вы всё испортили, верните как было!».

Важно хранить спокойствие и избегать поспешной реакции. Затаитесь по крайней мере на несколько дней. Напоминайте себе, зачем вы реализовали фичу и следите за тем, как и кому она помогает.

## **Не копите долг**

Велик соблазн отвечать на обратную связь обещаниями — мы вас услышали, обязательно поправим. Но это нарушает сразу два правила — «чистого листа» и «сырых идей». Обратная связь — это только сырье идеи, а не сформированные проекты. Самый правильный ответ чаще всего — вежливое «нет». Это не значит «совсем никогда» — возможно, вы решите вернуться

к предложению позже. Ответ «да», напротив, лишает вас этой свободы.

Помните — то, что вы только что выпустили, было результатом голосования и 6-недельного цикла. Если этой части продукта нужно больше времени — значит, нужен новый проект, формирование, презентация, голосование и реализация.

## **Сформированный проект, а не реакция на обратную связь**

Саму по себе обратную связь ещё нельзя отдавать в работу. Справедливо, когда обратная связь становится полноценным сформированным проектом и соревнуется с другими идеями на голосовании. Подробнее об этом мы говорили в главе 3.

Если отзыв действительно важный, посвятите следующие 6 недель формированию проекта, который гарантированно пройдёт голосование. Команды разработки в это время будут спокойно заниматься другими, уже сформированными, проектами.

# **Заключение**

---

## **Основные идеи**

Идеи, описанные в этой книге, тесно переплетены. Нужно пробовать на практике, какие именно решения лучше всего работают в вашей команде, и как конкретно их внедрить в текущие процессы.

Независимо от того, начнёте вы использовать метод Shape Up или нет, я надеюсь, что основные концепции дали вам пищу для размышлений:

- сформированный проект;
- аппетит, а не оценка времени;
- нужный уровень абстракции при формировании;
- наброски в виде макетных плат, наброски толстым маркером;
- правило предохранителя, передача команде всех полномочий по проекту;
- правильная длительность цикла (в нашем случае 6 недель);
- перерыв между циклами (в нашем случае 2 недели);
- разделение проекта на участки;
- движение вверх по холму (изучение) и вниз (исполнение);
- вколачивание объёма в сроки

## **Пишите нам**

Мы будем рады любой обратной связи, которая поможет сделать метод легче во внедрении. Вам кажется, что

мы упустили что-то важное? Что-то всё ещё неясно?

Конечно, мы также будем рады вашим рассказам об успешной (или провальной) практике использования метода в вашей компании.

Пишите: [shapeup@basecamp.com](mailto:shapeup@basecamp.com)

# Как работать в Basecamp по методу Shape Up

---

Мы создали Basecamp, чтобы работать в нём по методу Shape Up. Basecamp позволяет нам вести общение, управление задачами и документацию в одном месте. Вот как мы это делаем.

## Команда для формирования проектов

1. Создайте команду, которая будет формировать проекты. У нас она называется «Продуктовая стратегия».
2. Добавьте в команду людей, которые доверяют друг другу принципиальные решения — они будут давать отзыв на презентации проектов и отдавать свои голоса на голосованиях. Важно, чтобы команда была небольшой. Результаты работы этой группы публикуйте на всю компанию перед началом цикла.
3. Внутри команды, публикуйте презентации проектов в разделе Сообщений. Мы создали категорию «Презентации» с иконкой-лампочкой.
4. Используйте чат для обмена идеями и координации работы команды. Для самого голосования мы используем видео-звонок.

3.basecamp.com

Home Pings Hey! Activity My Stuff Find

Product Strategy

+ New message

## Message Board

All messages

**Group notifications**  
💡 Pitch by Ryan Singer • Feb 18 — Last November I wrote a pitch for adding Groups ala BCX to BC3: ↗ Groups! - Product Strategy. After looking at the pitch, 1

**Untangle the people spreadsheet**  
💡 Pitch by Ryan Singer • Oct 11, 2018 — Software gets complex when one thing does two things. Today the people spreadsheet is one interface that does at least 5

**Sort Messages Setting (revised)**  
💡 Pitch by Ryan Singer • Sep 21, 2018 — Here's a revised second take on the Sort Messages A-Z pitch. After talking to support and a couple customers via phone, 6

**Follow-up: Client-visible recordings**  
📝 Report by Ryan Singer • Sep 13, 2018 — In April we launched the new client-visible toggle to replace the Clientside. Do people use the new client toggle? How 2

**Creation on the iPhone**  
📝 Report by Ryan Singer • Sep 7, 2018 — While chatting w/ Conor today some questions came up about what gets created on the iPhone. Here's a rundown from 2

**Announcing Cycle 4**

FYI by Jason Fried  
Nov 17, 2017 · Notified 53 people

It's next cycle time!

First, this is a short cycle. We've got a few major holidays this cycle, plus the end of the year. People are busy with life, travel, snow (!), and all the other stuff that goes with ramping down one year and starting another.

So with that in mind, we're going to do something a bit different this cycle. We're only going to schedule a few core projects focused on making Basecamp feel simpler, a couple of experimental projects, and no dedicated small batch projects.

**Merge Latest Activity and Reports**  
Reports will move into the Latest Activity screen. When you click "Latest Activity" (or whatever we call it) in the nav bar, you'll see a series of report icons at the top, and "Latest Activity" will be the first one. So activity basically becomes a report like all the others, except that it's the first one you see automatically.  
Since we have a short cycle, and it's the end of the year, we're going to run a couple non-traditional projects.

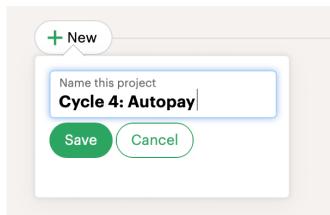
**Working prototype of interactive Hill Charts**  
We've been making Hill Charts manually in Apple Numbers and pasting them into Basecamp. Here's a great example of charts mapping the search-in-place project during last cycle.

This next cycle we're going to build an interactive prototype so we can do the mapping and dragging and positioning on a hill from right inside Basecamp. Scott is going to run the project and lead design. The goal is to have something we can use internally the cycle after next. If we continue to dig it, we'll release it as a full fledged feature and spend

## Проекты

1. Создайте Проект Basecamp для каждого проекта, над которым будет идти работа в течение цикла. Мы обычно добавляем номер цикла в название, например, «Cycle 4: Autopay.»
2. Добавьте в проект дизайнеров и разработчиков, которые будут над ним работать.
3. Опубликуйте сообщение с презентацией проекта (оригинальной или отредактированной для команды

разработки).



A screenshot of a project management interface. At the top, it shows a header with a person icon and the text "Cycle 4: Autopay". Below the header, there is a section titled "People on the project" with a "Add people..." button. The main content area is titled "1. Who do you want to add?". It lists two team members: Nicole Katz and Jared Davis. Each member has a row with their name, email, title, and company. There are "X" icons at the end of each row to remove the entry. At the bottom of the list, there are buttons for "Add another person" and "Pick people from a company...".

3.basecamp.com

Home Pings Hey! Activity My Stuff Find

BC3: Hill Charts > Message Board

## Hill Charts concept

Announcement by Ryan Singer Nov 17, 2017 - No one was notified.

Here's the starting concept for the Hill Chart feature in Basecamp.

### How they relate to to-dos

The basic idea is there's a hill chart at the top of the todoset. Hill **charts** map one-to-one with todosets, and the **dots** on the hill map one-to-one with todolists.

HILL ON TOP

TODO LISTS

3.basecamp.com

Home Pings Hey! Activity My Stuff Find

BC3: Hill Charts

## Campfire

Scott Upton 3:46pm We'll need to stub in the Hill Chart Editor, too (i.e. where you can annotate and update positions for each list)

Pratik Naik 4:07pm Scott I can add an empty controller for the editor. But I think it'd help to have some mocks first.

Scott Upton 4:15pm Cool, OK. I'll work something up tomorrow AM.

Pratik Naik 4:15pm Sounds good!

Thinking a little bit more about it, I think we may want to put the hill chart attributes in a new table and add one-one mapping with recordings. Similar to how the Positioning model works.

But we'll cross that bridge when we get there

For now, those attrs will live in the todolists table

## **Участки — списки задач**

1. По мере погружения в проект, команда создаёт задачи и размечает участки.
2. Внутри проекта, команда создаёт список задач для каждого участка, например, «Start Autopay» или «ACH Option.». Описание списка может содержать подробности, не вошедшие в название.
3. Команда добавляет все свои задачи в список, независимо от типа задачи (фронт-енд, бэк-енд, тестирование, прочее). Например, в примере ниже в списке «Start Autoplay» — две задачи для дизайнера и одна для разработчика. На странице задачи можно обсудить возникающие вопросы.
4. Повторяйте шаги 2 и 3 до готовности.

The screenshot shows a 'To-dos' list interface with the following details:

- Start Autopay**: 0/3 completed. Sub-tasks: Design button on the Invoice screen, UI for the Start Autopay screen, Hook into FI recurring billing API.
- Owner Shutoff**: 0/4 completed. Sub-tasks: UI for the owner to turn off autopay for a customer, Implement turning off, UI: What do they see after turning off?, Decide: Can they turn on again?
- ACH Option**: 0/0 completed. Sub-task: Add an ACH option to the Autopay form.

## Следите за прогрессом с помощью графиков-холмов

- На странице списка задач нажмите (•••) справа-сверху. Выберите Track this on the Hill Chart. Откроется страница с графиком, на котором появится точка, соответствующая текущему списку задач.
- Повторите для каждого списка задач, который нужно отслеживать.
- Чтобы обновить состояние графика, нажмите Update и перетащите точки. При необходимости добавьте комментарий.

4. Чтобы увидеть историю изменений, нажмите на дату сверху графика.

Cycle 4: Autopay > To-dos

0/3 completed

## Start Autopay

Set up Autopay with a credit card via the Invoice screen

- Design button on the Invoice to start 🚧 Jared D.
- UI for the Start Autopay screen 🚧 Jared D.
- Hook into FI recurring billing API 🚧 Nicole K.

Add a to-do

Edit Move Copy Notify me when new to-dos are added to this list Track this on the Hill Chart Add a group Archive

Cycle 4: Autopay

## To-dos 0/7

+ New list View as... ⚙️

Update

12 seconds ago

— ACH Option  
— Owner Shutoff  
— Start Autopay

0/3 completed

## Start Autopay

Set up Autopay with a credit card via the Invoice screen

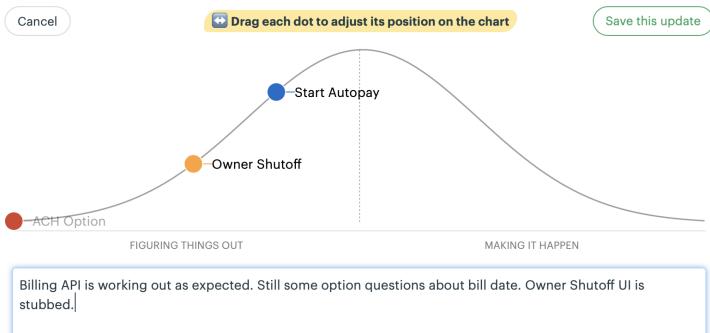
- Design button on the Invoice to start 🚧 Jared D.
- UI for the Start Autopay screen 🚧 Jared D.
- Hook into FI recurring billing API 🚧 Nicole K.

Add a to-do

0/4 completed

## Owner Shutoff

For v1, owners administrate Autopay on behalf of customers so we don't need t



Cycle 4: Autopay

+ New list View as...

## To-dos 2/9

Last updated 2 minutes ago

Update

Start Autopay

Owner Shutoff

ACH Option

FIGURING THINGS OUT      MAKING IT HAPPEN

2/5 completed

**Start Autopay**

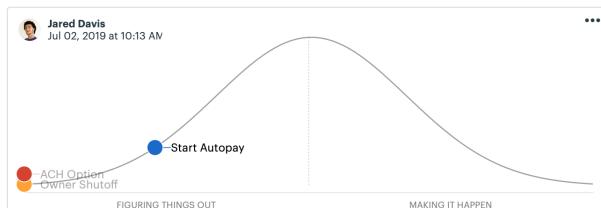
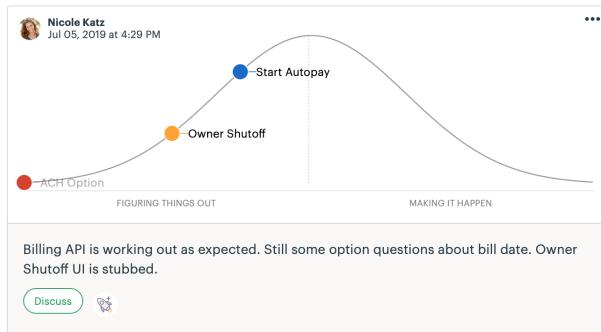
- Set up Autopay with a credit card via the Invoice screen
- Hook into FI recurring billing API
- Set correct bill date
- Move button to action menu on mobile

Add a to-do

- UI for the Start Autopay screen
- Design button on the Invoice to start

## Hill Chart Progress

Today



Basecamp включает в себя чат, сообщения, списки задач и документы — всё в одном интерфейсе, достаточно мощном для специалистов и достаточно простом для любого пользователя. Разработчикам, дизайнерам, тестировщикам и менеджерам одинаково уютно. Попробуйте бесплатно в течение 30 дней — <https://basecamp.com/>

# Отрегулируйте размер

---

## Общие правила и конкретные практики

Чтобы внедрение метода прошло быстрее и лучше, отделите общие правила метода от конкретных практик.

Формулировать работу — тоже работа. Это — формирование проекта. Формирование устанавливает границы и ожидания для тех, что будет реализовывать задуманное. Если не определить, что конкретно нужно делать, а чего делать не нужно, команде придётся принимать эти решения в спешке, под давлением сроков и технических ограничений.

То же с длиной цикла. 6 недель может быть для ваших команд много или мало. Но последствия работы вообще без чётких адекватных сроков одинаковы для всех. Независимо от длины цикла, передача полномочий и правило предохранителя обязательны.

На этапе разработки будут обнаруживаться неопределённости, неважно, отслеживаете вы их на графике-холме или нет. Обязательно отделить то, что известно от того, что неизвестно, и брать задачи в работу исходя из этого.

Эти общие правила не зависят от размера компании. Однако конкретные практики могут меняться. Рассмотрим два примера — крошечный стартап и большая корпорация.

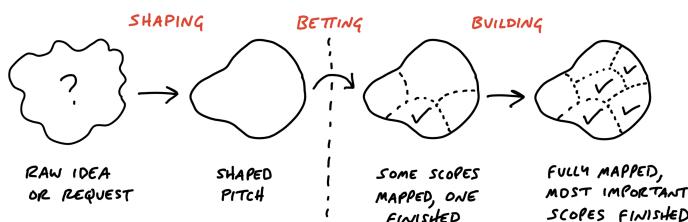
## Пример 1: крошечный стартап

Когда в компании 2-3 человека, каждый делает всего понемногу. Из-за этого трудно посвящать длительное время конкретным проектам, не отвлекаясь. Например, разработчик должен заниматься поддержкой и оборудованием, и всё это одновременно.

В маленькой компании проще общаться и менять направление. Встретились в чате или в коридоре — и вот вся компания в курсе дела.

Поэтому всё, что касается структуры, можно выбросить. Вам не нужны 6-недельные циклы, перерывы, формальные презентации и голосование. Вместо команд менеджеров (формирующих проекты) и разработчиков (реализующих проекты), одни и те же люди делают и то, и другое.

Каждый раз решайте по ситуации, что дальше. Сформируйте проект, оцените аппетит, реализуйте его, беритесь за следующий. Аппетиты могут быть разные — то две недели, то три. Вы всё делаете по методу Shape Up, просто более гибко, без жёсткой структуры и этапов.



## Пример 2: большая корпорация

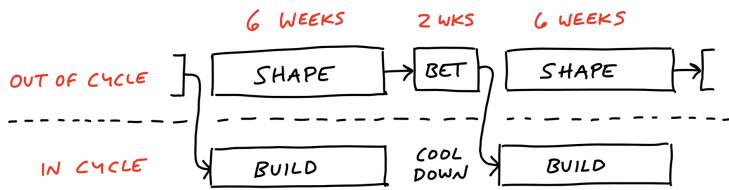
По мере роста компании, гибкость становится недостатком. Спонтанные встречи в чате уже не работают. Координация начинает занимать больше времени, чем сама работа. Кто-нибудь всё время что-нибудь забывает.

В случае большой компании стоит придерживаться строгой структуры, описанной в книге. Чем больше людей реализуют проекты, тем больше людей должно их должным образом формировать.

При текущем размере компании Basecamp (около 50 людей всего, около 12 занимается продуктом) мы можем держать отдельные команды дизайнеров и разработчиков, которые работают над своими проектами без необходимости отвлекаться на другие задачи.

Отдельная команда SIP (Security, Infrastructure, and Performance) занимается инфраструктурой. ИТ-команда отвечает за работоспособность сервисов. Технические специалисты поддержки общаются с пользователями. Всё для того, чтобы команды разработки могли полностью погрузиться в работу над проектами.

Если размер команды позволяет, формировать проекты и реализовывать их можно параллельно. Проект, сформированный и одобренный на текущем цикле, пойдёт в разработку в следующем цикле.



# С чего начать

---

## Вариант 1: эксперимент в 6 недель

Не обязательно менять все процессы сразу. Если компания не готова к глобальному изменению, проведите эксперимент длительностью в 6 недель. Вот из чего он состоит:

1. Сформируйте проект, достаточно значимый для компании, который реально закончить за 6 недель. Будьте пессимистичны и добавьте немного времени на первый раз.
2. Заберите на проект одного дизайнера и одного разработчика. Добейтесь гарантии, что никто не будет их отвлекать во время эксперимента.
3. Вместо голосования, просто познакомьте команду со сформированным проектом. Важно сделать полноценную презентацию, со всеми ингредиентами, упомянутыми ранее. Познакомьте их с методом, договоритесь, что они сами обнаруживают и создают себе задачи.
4. Организуйте для команды единое рабочее место (реальную комнату или чат).
5. Мотивируйте команду добиться готовности их первого участка.

На этом этапе не стоит тратить силы на карту участков или подробные *отчёты о прогрессе*. Наверняка вы заметите прогресс уже потому, что отдали в работу хорошо сформированный проект, защитили команду от отвлекающих факторов и предоставили им свободу принятия решений.

Используйте успех эксперимента как аргумент для более полномасштабного внедрения.

## **Вариант 2: сначала только формирование**

Иногда нет возможности получить команду на 6 недель. Тогда можно начать внедрение, ограничившись первым этапом — формированием проекта. Воспользуйтесь методом Shape Up, чтобы сформировать проект, затем презентуйте его и пустите по принятому в вашей компании процессу (даже если это означает напихать задачи в беклог, как в шреддер).

Качественно сформированный проект может убедить коллег попробовать и другие практики.

## **Вариант 3: сначала только циклы**

Ещё можно начать работать циклами. Команды, которые привыкли к двухнедельным спринтам, испытают облегчение, избавившись от постоянного планирования и получив возможность «поймать волну».

## **Результат важнее плана**

Прокачайте свои навыки достижения результата прежде, чем навыки формирования и исследования. У вас может быть

лучшая в мире идея, но она не имеет смысла, если вы не можете превратить её в реализованный проект. Дайте команде почувствовать, что она может доводить дело до конца, и уже потом можно заниматься улучшением исходных материалов.

## **Результат важнее всего**

Часто страшно давать командам свободу решать, что делать по проекту. Что если они неграмотно распорядятся временем? Что если дизайнер или разработчик окажутся в какой-то момент без задач?

Чтобы преодолеть этот барьер, обратите внимание не на микро-уровень, а на макро. Спросите себя — если проект будет успешно завершён в течение 6 недель, будет ли это означать, что мы всё сделали правильно? Если проект завершается, цели достигаются и у команды есть ощущение прогресса — является ли это успехом? Если да, то не нужно управлять каждым часом каждого работника. Результат важнее всего.

# Термины

---

## **Аппетит (Appetite)**

Количество времени, которое мы готовы потратить на проект (не путать с оценкой времени каждой задачи)

## **База сравнения (Baseline)**

Текущая реальность — как сейчас пользователи решают проблему, пока мы реализуем проект.

## **Голос (Bet)**

Решение отдать команде в работу проект, с обязательством не отвлекать её и с ожиданием получить завершённый проект в конце цикла.

## **Голосование (Betting table)**

Встреча во время перерыва между циклами, участники которой обсуждают презентации и голосуют за проекты для следующего цикла.

## **Большой проект (Big batch)**

Проект, для реализации которого нужен ресурс одной команды на один цикл.

## **Печатная плата (Breadboard)**

Набросок интерфейса, который показывает сущности и их связи, но не внешний вид и свойства.

## **Правило предохранителя (Circuit breaker)**

Приём управления риском: проекты, которые не завершены к концу цикла, отменяются, а не продлеваются.

### **Чистка (Cleanup mode)**

Последний этап работы над новым продуктом, во время которого проекты не формируются, нет чётких команд, результаты публикуются по мере готовности.

### **Перерыв (Cool-down)**

Двухнедельный период между циклами для отдыха, исправления багов, планирования, обсуждений.

### **Цикл (Cycle)**

Шестинедельный период для непрерывной работы над проектами.

### **Снижение риска (De-risk)**

Увеличение вероятности завершения проекта в срок за счёт обнаружения и устранения белых пятен.

### **Обнаруженные задачи (Discovered tasks)**

Задачи, созданные командой по мере работы над проектом, а не заранее.

### **Путь вниз по холму (Downhill)**

Статус задачи, участка или проекта, при котором все неопределённости сняты и остаётся только выполнение.

### **Набросок толстым маркером (Fat marker sketch)**

Набросок интерфейсного решения, специально нарисованный толстой линией, чтобы не прорисовывать детали.

### **График-холм (Hill chart)**

График, показывающий состояние работы на графике с отметками «неясно», «ясно», «сделано».

## **Айсберг (Iceberg)**

Участок проекта, в котором бэк-энд значительно сложнее фронт-енда или наоборот.

## **Придуманные задачи (Imagined tasks)**

Задачи, которые команда создаёт ещё до начала работы над проектом, только ознакомившись с ним. См. Обнаруженные задачи.

## **Торт (Layer cake)**

Участок проекта, в котором сложность бэк-енда и фронт-енда примерно одинакова, и можно оценить работу по бэк-енду, глядя на интерфейс.

## **Уровень абстракции (Level of abstraction)**

Количество деталей, которые мы используем в описании проблемы или решения.

## **Обязательные задачи (Must-haves)**

Задачи, которые нужно завершить, чтобы весь проект считался завершённым.

## **Необязательные задачи (Nice-to-haves)**

Задачи, которые можно не делать, если на них не хватает времени. Они не влияют на завершённость проекта и обозначаются знаком «~» в начале названия.

## **Презентация (Pitch)**

Документ, описывающий сформированный проект для обсуждения и голосования.

## **Режим разработки (Production mode)**

Второй этап создания нового продукта, после Исследования.

Принципиальные архитектурные решения приняты, работа идёт по стандартному методу Shape Up.

### **Белое пятно (Rabbit hole)**

Часть проекта, содержащая неопределенность, имеющая неизвестную сложность, или пробелы в формулировках.

### **Режим исследования (R&D mode)**

Первый этап создания нового продукта, во время которого продукт берёт в работу руководство, цель — разобраться, а не разработать.

### **Сырая идея (Raw idea)**

Запрос или предложение, выраженное словами и не оформленное как полноценный проект.

### **Участки (Scopes)**

Части проекта, которые могут быть описаны, проработаны и завершены независимо друг от друга.

### **Вколачивание объёма работы в срок (Scope hammering)**

Постоянное подвергание задач и решений сомнению, с целью найти и отменить работу, без которой можно обойтись.

### **Сформировать (Shape)**

Сделать абстрактную (сырую) идею более конкретной — придумать решение проблемы и описать его ключевые элементы.

### **Шесть недель, горизонт дедлайна (Six weeks, Time horizon)**

Стандартный размер цикла. Достаточно много, чтобы разработать что-то осмысленное, и достаточно мало, чтобы видеть финиш уже на старте.

### **Набор мелких проектов (Small batch)**

Несколько проектов в 1-2 недели каждый, которые команда должна завершить к концу 6-недельного цикла, в любом порядке.

### **Путь вверх по холму (Uphill)**

Статус задачи, участка или проекта, при котором выясняют, что именно нужно сделать, и как конкретно это сделать.

## Об авторе

---

Райан Сингер (Ryan Singer) прошёл все уровни разработки, от дизайна интерфейсов и бэк-енда до продуктовой стратегии.

За 17 лет работы в Basecamp он реализовал множество проектов, которыми пользуются миллионы людей, и создал процессы, с помощью которых команды придумывают и реализуют то, что нужно.

Сегодня Райан работает над стратегией. Его работа — понять, какие проблемы решают пользователи Basecamp и как изменить продукт, чтобы он лучше им в этом помогал.

# Предисловие

---

От того, как команда организует работу, во многом зависит то, чего она достигнет. Процессы, методы, практики, подходы, дисциплина, стиль общения, настрой. Ответ на вопрос «как?» — ключ к успеху любого дела.

Часто говорят: «Результат важнее всего». Но это не всегда так. Точнее, почти всегда не так.

Если говорить о разработке программных продуктов, достижение результата не тем путём часто приводит команду к выгоранию, недоверию, и в итоге к развалу. Да, работа сделана, но какой ценой? Как мы выдержали всё это? Неужели нужно проходить через те же мучения снова и снова, год за годом?

Посчитайте, сколько у вас было проектов, за которые вы с радостью взялись бы снова? И наоборот, на скольких проектах сроки были сорваны, люди выгорели, ожидания руководства не имели ничего общего с реальностью? Сколько раз люди ссорились, разочаровывались и уходили?

Да, иногда результат — «всё». Совсем «всё».

Тогда какой путь — правильный?

Мы в Basecamp часто замечаем интерес к тому, как организована наша работа. Люди удивляются — как у нас получается делать так много, так быстро, такой небольшой командой. И почему никто не выгорает и не увольняется.

Во-первых, мы не адепты Waterfall, Agile или Scrum. Во-вторых, мы не обклеиваем стены заметками. В-третьих, мы не проводим ежедневные стендапы, спринты, и вообще ничего, что хотя бы отдалённо предполагает работу на износ. Никаких бэклогов, Канбан-досок, расчётов личной эффективности, ничего такого.

Мы описывали некоторые из наших методов в блогах, мастер-классах и конференциях, но не сводили их в единую систему. В этой книге свели.

Мы подробно описали наши рабочие процессы и готовы поделиться ими со всеми, кому они интересны. С теми, кто неравнодушен к тому, как ещё может быть, кто хочет работать лучше, чем сейчас.

Эта книга — маяк. Ваша команда достаточно блуждала в темноте. Мы предлагаем новую дорогу и надеемся, путешествие вам не только понравится, но и принесёт реальную пользу.

Приятного чтения.

