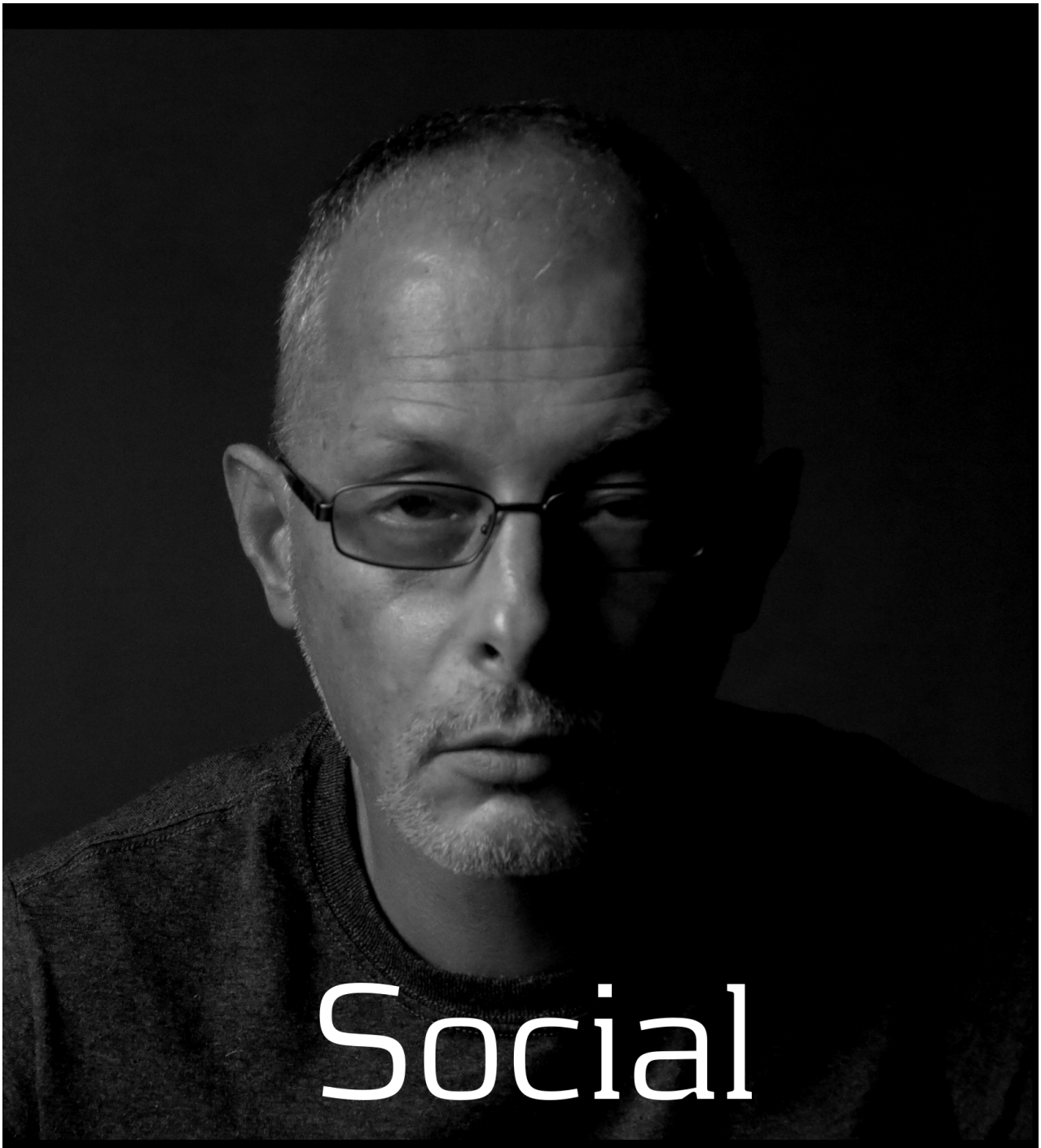
Социальная Архитектура

Создание онлайн сообществ

Питер Хинченс

Version #2c3a68eb8b81d8fb749313eacf826260e99f2052, 2020-06-02

Содержание



Social Architecture

Building On-line Communities

by

Pieter Hintjens

- Читать книгу в браузере
- Скачать Epub
- Репозиторий на Github

Посвящается

Моим друзьям

Предисловие

Мудрость толпы

В «Рассуждениях о первой декаде Тита Ливия» Никколо Макиавелли есть следующие строки:

> «Что же до рассудительности и постоянства, то уверяю вас, что народ постояннее и много рассудительнее всякого государя. Не без причин голос народа сравнивается с гласом Божьим: в своих предсказаниях общественное мнение достигает таких поразительных результатов, что кажется, будто народ ясно предвидит».

В своей книге «Мудрость толпы» Джеймс Шуровьески писал: «при правильных условиях группы могут быть очень умными, а зачастую могут быть намного умнее, чем даже самый умный человек внутри группы». Он заметил, что коллективный разум обычно показывает лучшие результаты, чем небольшая группа экспертов, даже если члены группы не владеют всеми фактами или ведут себя иррационально, поступая по-своему.

Другими словами, группа случайных людей в среднем будет умнее нескольких экспертов. Этот тезис противоречит здравому смыслу и выглядит насмешкой над накопленной веками мудростью. Эксперты в области человеческого интеллекта (социологи, антропологи, психологи) встретили идеи Шуровьески далеко не с распростертыми объятиями. Он пошел дальше: «добавив в группу специалистов, вы сделаете ее глупее, а добавив дилетантов, повысите опять ее интеллектуальный уровень. Как и любой рецепт, это работает только при определенных обстоятельствах».

Я наткнулся на Шуровьески, когда стал работать над универсальным рецептом по построению сообществ. И сразу увидел, что его работа тесно перекликается с моими собственными наблюдениями, и что я могу устроить ей испытание. У меня была возможность не только применить, но и обкатать его идеи на многих сообществах и поискать им опровержение: все по-научному.

Это намерение дало начало процессу по построению умных, самоуправляемых, успешных онлайн-сообществ, которые раз за разом опережали экспертные группы. Эту дисциплину я назвал Социальной Архитектурой, что позволяло мне некоторое время называть себя «Социальным Архитектором» (сегодня я — пытающийся пробиться писатель, это звучит романтичней).

Социальная Архитектура по аналогии с привычной архитектурой является процессом и результатом планирования, разработки и становления онлайн-сообщества. Социальная архитектура в виде онлайн-сообществ — это культурные и политические символы и произведения искусства цифрового общества. 21 век будет ознаменован зарождением Социальных Архитекторов.

Успешные онлайн-сообщества обычно основаны на договоренности о взаимной выгоде, подразумеваемой или явной. Т.е. это возможность построить бизнес на миллиард долларов,

основанный на добровольном труде участников, действующих из эгоистических побуждений. Часто участники не осознают или не придают значения тому, что они являются частью сообщества. Однако подоплека любого нашего действия — получение выгоды. «Краудсорсинг» является эксплуатацией добровольного труда для получения прибыли. И это работает только в том случае, когда толпа на самом деле хочет решить проблему, которую вы подбросили, или которую она обнаружила.

Мудрее и постояннее всякого государя

Макиавелли не дал объяснений или подтверждений своему наблюдению. Однако понимание того, что коллективная воля является безошибочной и справедливой — vox populi, vox Dei — пронизывает современную культуру. Она поддерживает в нас веру в демократию и оправдывает наши требования прозрачности и доступа к информации. Это основа современной экономики, составляющие части которой — свобода выбора и торговли.

Шуровьески определил четыре элемента, необходимых для умной толпы: разнообразие мнений, независимость членов друг от друга, децентрализация и эффективные способы агрегировать мнения. Он описывает образцовую умную толпу как состоящую из многих независимо мыслящих индивидуумов, которые тесно связаны, которые географически и социально разделены, которые беспристрастны к предмету, каждому из которых доступны многие источники информации и которые имеют некую возможность объединить свои индивидуальные суждения в одно коллективное решение.

Согласно Шуровьески умные толпы выносят более точные и быстрые решения, самоорганизуются для лучшего использования ресурсов и сотрудничают без центральной власти. Некоторые примеры умных толп, такие как Википедия, чрезвычайно успешны, несмотря на интенсивную и непрекращающуюся критику со стороны негативистов и атак вандалов и конкурентов. Эти идеи настолько захватывающие, что остается только гадать, почему мы не наблюдаем все больше и больше умных толп. На самом деле, почему мир ширится глупостью, когда мудрость так возможна, так близка?

Есть много хороших объяснений для глупости многих толп, и я исследую детально этот вопрос в своей книге «Культура и Империя», откуда была взята эта часть. Мало кто пытался объяснить глупость толпы с точки зрения коллективного интеллекта. А без ясного понимания правильного действия, как мы можем надеяться объяснить неправильное?

Поэтому очевидная неудача коллективного разума убеждает многих в том, что это лишь забавная теория, которая не применима на практике. И все же если мы посмотрим на онлайн-сообщества, например на те, которые формируются вокруг популярного open-source проекта программного обеспечения вроде ZeroMQ, мы увидим группы, очень похожие на описанные толпы Шуровьески. И пока, может быть, сложно опознать умные толпы в реальной жизни, кажется, что они являются доминирующей моделью онлайн. Пробы и ошибки позволили цифровому обществу заново открыть принципы умной толпы и перенять их в качестве своих базовых принципов действия.

Решение цифрового общества древней проблемы коррумпированной власти элегантно и успешно. Существуют буквально миллионы сообществ, каждое из которых полагается на власть своих основателей. Граждане цифрового общества свободно выбирают, какие власти

уважать и какие игнорировать. Основной фокус заключается в принятии власти без наделения ее «правом» командовать.

К тому же возникает острая конкуренция по созданию справедливой власти, которая бы не командовала, а принуждала соблюдать необходимые правила. Это взрывоопасная истина. Поколения, которые познакомятся с этой моделью, не согласятся — даже под угрозой смерти — уважать модель индустриального общества, где, при необходимости, для убеждения могут использоваться железные занавесы и вооруженные пограничники, где граждане буквально принадлежат Государству.

Истоки Социальной Архитектуры

Я поставил кучу денег на Социальную Архитектуру и получил приличную выгоду. Она близка к строгим социальным научным дисциплинам, на стороне которых годы воспроизводимых экспериментов на реальных случаях и исследований существующих сообществ. В ней смешиваются психология, экономика, политология, технология, гуманизм и оптимизм во что-то такое, что, как я обнаружил, может сделать многих людей счастливыми.

Мое путешествие в область Социальной Архитектуры началось в конце 1990-х годов с изучения книги о том, как культы эксплуатируют наши социальные инстинкты. Секты — место не из приятных, конечно. Однако людей в них затягивает потому, что мы все — социальные животные, которые последний миллион лет ради выживания развивали инстинкт объединения и образования групп. Готовность уважать власть, подчиняться правилам, изучать общие языки и адаптироваться под общее поведение стало для нас второй натурой. Секты подвергают своих членов идеологической обработке, играя на этих инстинктах. Они разделяют членов с их семьями, исключают уединение, перегружают жаргоном, создают деспотичное правление, беспорядочно наказывают и награждают.

Подобным образом секты могут превратить большинство обычных людей в бездумных последователей, которые добровольно опустошат свои банковские счета, украдут у своих родных и будут работать годами, не требуя платы. Еще студентом, наблюдая исчезновения случайных друзей в подземельях Сайентологии и других культов, все это представлялось мне чем-то злокачественным и необъяснимым. Позже, когда мой ближайший кузен выпал из жизни и потратил пять лет своей жизни на Сайентологию, для меня эта тема стала личной.

Изучая материалы сайта Cult Information Centre (CIC), меня вдруг осенило, что все эти техники по промывке мозгов имеют несколько общих черт. Во-первых, они определенно нацелены на уничтожение того, что делает нас сильными — они атакуют независимое мышление и поведение. Во-вторых, они напоминают мне обстановку, в которой мне уже приходилось работать (крупный бизнес часто похож в своей деятельности на секту). В третьих, казалось, что все они реверсивные, т.е. их можно использовать в обратную сторону для благих дел.

Последнее наблюдение необычно. Если молоток разбивает окно, то вряд ли что-то изменится, если вы перевернете молоток. Но на некоторых примерах это наглядно видно. Вот одна из техник с сайта CIC: «Социальное давление в группе: подавляйте сомнение и сопротивление новым идеям, играя на потребности в принадлежности к группе». Реверс —

снижение стоимости присоединения к группе и выхода из нее будет способствовать возникновению новых идей и критики». Или вот: «Исключение уединения — способность давать логическую оценку будет снижаться при отсутствии возможности размышлять наедине». Ее реверс: «предоставьте людям пространство и время для уединения, и они станут логичней размышлять».

Мои умозаключения — стойкие. Мы выживаем благодаря присоединению к группам, следованию за другими и попыткам понять мир. Некоторые группы существуют за счет одомашнивания нас и низведения до уровня животных. Другие — дарят нам свободу и позволяют стать сильнее, умнее и независимей.

В 2000 г. интернет еще не стал достаточно дешевым для масс, и open-source сообщества были маленькими, региональными, сосредоточенными вокруг университетов. Такие open-source сообщества как Debian Foundation до сих пор функционируют как классические некоммерческие организации, как легальные юридические лица с советом директоров, казначеями и пр.

В 2005 г. я стал участником ряда совместных проектов. С одной стороны, я был вовлечен в проект FFII, направленный на борьбу с патентами на программное обеспечение в Европе. Мы (хорошие парни) выступали в европейском парламенте, спорили с Европейским патентным офисом (плохие парни), организовывали семинары, предлагали поправки, собирали голоса и, вообще говоря, участвовали в сильнейшем лоббировании, которому когда-либо подвергался Брюссель.

С другой стороны, я занимался разработкой открытых стандартов, начиная с Advanced Message Queuing Protocol (AMQP). Культурный контраст между этими двумя организациями был очень сильным. FFII была группой безумных добровольцев, невероятно креативных, преисполненных холодной, жесткой решительностью остановить SAP, Siemens, Microsoft и Nokia (еще более плохие парни) в их стремлении изменить европейское законодательство с целью легализации серого рынка патентов на программное обеспечение. В рабочую группу AMQP входили банки и крупные компании-разработчики программного обеспечения, которые тоже оказались по-своему безумны, и при этом более неприглядны.

Оказавшись окруженным со всех сторон безумием, я вдруг опять подумал о важности исследований социальных инстинктов и сектантских техник. С моими друзьями из FFII мы запускали компанию за компанией. Сайты, петиции, рассылки по эл. почте, конференции… этому не было конца. Большинство из наших компаний не достигали достойного масштаба, лишь некоторые из них. Но что важнее, в течение трех лет мы экспериментировали и собирали информацию.

Мы поняли две важные вещи. Во-первых, культ является обратной стороной умной толпы. Сектантские паттерны казались отточенными, и я наблюдал, как одни люди применяют их по отношению к другим людям снова и снова. В любой тесной группе, семье, компании или команде начинают проявляться черты культа, в большей или меньшей степени. Все дело в градусе. Однако, как только вы тратите ваше свободное время на чей-то проект, вы существенно начинаете скользить вниз с этого склона. Я видел, как целые группы сходили с рельс и не могли больше думать трезво или выдавать точные результаты. Наблюдалась четкая причинно-следственная связь: чем больше группа становилась похожа на секту, тем более бесполезной она становилась.

Во-вторых, просто реверсировать сектантские техники не достаточно. Да, это помогает с самого начала развивать личную креативность и силу, но это не то же самое, что создание крепкого сообщества. Для этого вам требуются более конкретные паттерны. Определите убедительную миссию, чтобы привлечь новичков. Сделайте так, чтобы людям было легче начинать. Приветствуйте споры и конфликты, ведь в них рождаются хорошие идеи. Систематически делегируйте полномочия, создавайте соперничество. Работайте больше с добровольцами, чем с наемными сотрудниками. Добейтесь разнообразия и размаха. Пусть люди владеют работой, а не работа — людьми.

Конечно, намного дешевле и быстрее проводить крупные эксперименты с людьми онлайн, чем в реальном мире. Для подтверждения или опровержения рецепта по построению сообщества, все, что вам нужно сделать, так это создать пространство, определить некоторые правила игры, объявить об этом миру, откинуться на спинку кресла и ждать.

Мой самый крупный и наиболее успешный эксперимент на сегодня, к которому я буду часто обращаться — это сообщество программного обеспечения ZeroMQ. Оно выросло из команды, собиравшейся на одном из чердаков Словакии, в мировое сообщество, и оно используется тысячами организаций. Кроме того, ZeroMQ было полностью создано и руководилось своим сообществом: более ста создателей корневой библиотеки, и более ста связанных с этим проектов.

Chapter 1. Инструментарий

Мой инструментарий социального архитектора состоит из 20 инструментов, каждый из которых соответствует какому-либо аспекту сообщества или группы. Их можно использовать двумя способами.

Во-первых, с их помощью вы можете делать **измерения существующего сообщества**, оценивая его по шкале от нуля и выше.

Во-вторых, вы можете использовать их для **создания сообщества**, при этом прилагая усилия там, где они наиболее необходимы.

- Четкая миссия – заявленная причина существования группы.
- Свободное участие – насколько легко люди могут присоединиться к группе.
- Прозрачность– насколько открыто и публично принимаются решения.
- Бесплатные участники – как много можно платить людям за участие.
- Свобода работы с материалами (ремиксабельность) – насколько свободно участники могут использовать работу друг друга.
- Четкость протокола – насколько хорошо прописаны правила.
- Компетентность власти – насколько хорошо следят за соблюдением правил.
- Нон-трайбализм – насколько далеко распространяются права группы над своими участниками.
- Самоорганизация – насколько свободно могут участники определять свои задачи.
- Толерантность – как группа разбирается с конфликтами.
- Измеримый успех – как хорошо группа может отслеживать свой прогресс.
- Высокое награждение – как группа вознаграждает своих участников.
- Децентрализация – насколько широко распределены участники группы.
- Свободная рабочая среда – насколько легко создавать новые проекты.
- Стандартная структура – насколько общая структура стабильна и предсказуема.
- Плавность обучения – насколько легко начать и продолжить учиться.
- Позитивность – насколько группа движима позитивными целями.
- Чувство юмора – насколько серьезно группа себя воспринимает.
- Минимализм – сколько лишней работы делает группа.
- Разумное финансирование – как группа борется за выживание в экономическом плане.

NOTE	Спасибо Сергею Даньшину за помощь с переводом.

Мы рассмотрим эти инструменты поочередно и увидим, как они работают в разных сообществах. Для начала несколько общих советов о создании сообщества. Будьте предельно честны с собой и с другими. Главное для вас – это преодолеть собственные

предубеждения и пристрастия, а уже потом – те, что присущи вашим коллегам.

Каким бы инструментом я вас ни снабдил, вы захотите адаптировать его и подогнать к собственным нуждам. Социальная архитектура является все еще молодой наукой, и многие из моих инструментов будут слишком громоздкими или неполными.

Вот лучший способ, как поступить:

- Пользуйтесь вашим собственным продуктом. Если вы не будете фанатичным пользователем продукции вашей группы, считайте себя полуслепым. Я осознал это, когда работал в компании-производителе пива Nigerian Breweries в 1990-х годах: наслаждаясь пивом, я научился оценивать бизнес по продаже пива намного глубже.
- Практикуйтесь и повторяйте пройденное. Экспериментировать дешево, а неудачи неизбежны. На самом деле, если вы начнете проект, а он не удастся, то никто и не заметит. Поэтому начинайте много проектов и меняйте или отлаживайте ваши инструменты, если они не работают. Займитесь оперативной поддержкой пользователей. Во всех сообществах есть места, куда обращаются новички, чтобы задать вопрос. Будьте там, наблюдайте, почему новопришедшие теряются, какие делают ошибки, и на основе этого вносите соответствующие правки в вашу разработку. Возможно, они заблуждаются насчет основного предназначения. Или, может, им непонятна структура. Хороший разработчик симпатизирует своим пользователям, сопереживает и помогает им.
- Выпускайте релиз сразу и почаще. Это мантра разработчиков свободного ПО. Это очень правильно. Вы хотите вести разработку в открытую и получать критические отзывы, и чем раньше, тем лучше. Мы в ZeroMQ выпускали патчи, как только они были готовы.
- Учитесь и преподавайте постоянно. Обучение других позволяет видеть перспективу, а изучение нового самому позволит в дальнейшем использовать новые инструменты. Социальная архитектура – молодая дисциплина, и хотя ее корни уходят вглубь человеческой психологии, перед ней еще стоит много вопросов.

1.1. Четкая миссия

Отправная точка создания любого сообщества – формулировка его миссии. Она определяет цели, которые мы разделяем, еще до того, как присоединиться к проекту. Это как заголовок сайта или рекламный лозунг фильма. Например, заголовок у Reddit звучит так: «главная страница интернета» – амбициозная миссия, которая, тем не менее, выполнена. Слоган Фейсбука: «помогает вам связаться и поделиться с людьми в вашей жизни».

СОВЕТ: используйте вашу миссию в качестве слогана на сайте, в рекламе, в презентациях и т.д. Если вы инвестируете деньги в ваше сообщество, подумайте о том, чтобы зарегистрировать формулировку миссии как торговую марку.

Без четкой миссии онлайн-сообщество не будет расти. Друзья, которые начали заниматься проектом, могут быть все согласны с тем, чего они хотят достичь, но каждому новичку придется гадать, что они имеют в виду. Люди будут заблуждаться в своих догадках и со временем могут изменить свое мнение. Будут возникать разногласия, путаница и разочарование по мере понимания людьми того, что их тяжелый труд был впустую, т.к.

остальная часть группы движется в другом направлении.

Реакция людей на миссию компании не должна быть «да, это благоразумно», а наоборот: «вы же это не серьезно, так?!». Миссия Википедии – "«бесплатная энциклопедия, которую любой может редактировать»", – хороший тому пример. Это было изначальной целью, поэтому все прочие, кроме считанных идеалистов, отнеслись к этому как к чему-то невозможному и бредовому. На это и был расчет Википедии: чтобы эти идеалисты однажды взошли на борт. Невозможные цели привлекают правильных людей к молодому проекту.

СОВЕТ: Меняйте миссию по мере взросления вашего сообщества. Вначале вы захотите привлечь молодых идеалистов и первопроходцев, потом – лидеров, а потом – первых адептов, широкую общественность, дальнейших последователей. Каждая из этих групп стремится к разному. Поняв это, соответственно измените миссию.

Чтобы сформулировать хорошую миссию, ориентируйтесь на одну основную проблему, которой посвящен ваш проект. Reddit, например, решает проблему того, как получить новости из интернета при слишком большом количестве ресурсов с интересной информацией. Его «основная страница» представляет собой цифровую газету 21-го века. Википедия решает проблему аккумулирования знаний миллионом умов. Слова «любой может редактировать» – так же, как и «Глас народа – глас Божий», – говорят о том, что если и возможно найти истину, то только сообща.

СОВЕТ: При намерении сделать что-либо, много или мало, всегда старайтесь начать с определения проблемы, которую вы хотите устранить. Только когда у вас будет четкая и реальная проблема, с наличием которой все согласны, только тогда приступайте к обсуждению возможных ее решений. Решение вымышленной проблемы схоже с группой без четкой миссии. У вас может быть несколько миссий, случайно или преднамеренно. Если миссии тянут сообщество в разных направлениях, это может кончиться плохо. Например, рост группы может потребовать вложений, что вступит в противоречие с позицией по вопросу прибыли. Если бы Википедия стала коммерческой организацией, с рекламой и комплектом высокооплачиваемых менеджеров, то это, по-вашему, привело бы к ее расширению или упадку?

В случае с ZeroMQ наша миссия звучала так: «Быстрейшая. Передача сообщений. Всегда». Это хорошее и почти невозможное решение проблемы, насчет существования которой мы все соглашались, а именно: доступные на тот момент технологии были медленными и неповоротливыми. В то же время, мы с моим партнером-основателем Мартином расходились в целях. Он хотел создать лучший из возможных программных продуктов, я же хотел создать крупнейшее из возможных сообществ. По мере того, как росло число пользователей, его драматическое изменение, что сломало существующие приложения, причинило усиливающуюся боль.

В данном случае мы смогли сделать всех счастливыми (Мартин ушел, чтобы работать над созданием новой библиотеки, названной «Nano»). Тем не менее, если вы не можете разрешить противоречия, касающиеся миссии, они могут серьезно навредить проекту. Проекты могут вынести многие споры, а вот разногласия между основателями довольно травмоопасны.

СОВЕТ: Если основатели согласны, что «успех» определяется как «максимально возможное

сло участников», то в последующие годы это может помочь в сохранении целеустремленности. Это также облегчает измерение вашего успеха по мере развития.

1.2. Свободное участие

Определившись с миссией, вам нужно протестировать ее в реальном мире. Это значит, вам нужно дать краткий, но убедительный ответ на ту проблему, на которую вы нацелились. Я называю это «посевом». Этот процесс преследует две основные цели. Во-первых, начать собирать идеалистов и первопроходцев (в основном тех, кто был настолько безумен, чтобы поверить вам) в сообщество. Во-вторых, доказать или опровергнуть вашу миссию.

Проекты могут кончиться неудачей по многим причинам. Но главная причина — основополагающая идея или миссия были не настолько удивительными, как того ожидали люди. Неудача – нормально, даже отлично, если только она не стоила нескольких лет вашей жизни. Посадить семечко и показать его только нескольким людям не достаточно, потому что большинство людей не будет критиковать. Из жалости. Однако попросите их потратить хотя бы несколько часов своего времени на то, чтобы сделать проект лучше, и если они не скажут «да», тогда вы поймете их настоящее отношение.

СОВЕТ: Привлеките к «посевному» проекту внимание публики и вдохновляйте людей присоединяться к нему с самого начала. Если люди вовлекаются в проект, скорее их продвигайте. А если этого не происходит, то считайте это знаком того, что ваша миссия может быть ложной. Используйте «посевной» проект, чтобы создать сообщество.

Когда люди соглашаются помогать вам, нужно обеспечить им место для совместной работы. Вам нужна «платформа для сотрудничества». Две моих самых любимых: Wikidot для информационных сообществ и GitHub для проектов по разработке ПО. Платформа должна быть бесплатной. С ней должно быть легко и в учебе и в работе. Ваш посевной проект должен быть виден анонимным участникам. Он должен работать для кого угодно, вне зависимости от его или ее возраста, пола, образования или местоположения.

Все это позволяет потенциальным заинтересовавшимся незнакомцам зайти и посмотреть на вашу работу, и если им она приглянется и они почувствуют в ней вызов, то смогут постепенно вовлекаться в проект. Вы хотите работать над вашим посевным проектом публично и говорить о вашем новом проекте с самого начала. Это значит, что люди смогут делать предложения и чувствовать вовлеченность с самого первого дня.

Если мы как основатели группы выбираем тех, с кем будем работать, мы создаем основание для предвзятого выбора. Намного легче работать с теми милыми, умными людьми, которые соглашаются с нами, чем с теми идиотами и критиками, которые выражают свое несогласие. А когда вы соглашаетесь со мной, вы подтверждаете все те мои иллюзии и допущения, которые, как я знаю по собственному опыту, могут оказаться ложными самым удивительным способом.

Со временем увеличение количества людей, которые разделяют те же неверные допущения и предубеждения, может привести к гибели проекта. Например, при разработке программных протоколов требования к крупным компаниями могут сильно отличаться от требований к маленьким open source командам. Поэтому если комитет по протоколу состоит полностью из крупных компаний, то результат их деятельности будет неприемлем

для массового рынка.

Решением является свободный доступ для всех заинтересованных, какой бы безумной и непохожей ни была бы их точка зрения. Это дает нам в перспективе широкое и разностороннее сообщество – предшественник умной толпы. В ZeroMQ мы никогда не отворачивались от тех, кто хотел участвовать. Я втягиваю людей, даже если их вклад в общее дело мал или неверен. Сообщество важнее, чем продукт.

Когда сообщество посевного продукта созреет, участники захотят создать его второе поколение. Как социальный архитектор, вы должны руководить этим так, чтобы усилия умной толпы были направлены на разработку «реального» продукта. Возможно, где-то на этом этапе вы захотите найти хорошее доменное имя и сделать «приличный» веб-сайт.

СОВЕТ: Если люди не присоединяются к вашему посевному проекту, не продолжайте заниматься им. Вместо этого разберитесь, что их останавливает, и устраните это. Начните заново с прополки. Не убивайте преждевременно побеги, людям требуется время, чтобы оценить то, что вы пытаетесь сделать.

1.3. Прозрачность

Прозрачность очень важна для быстрого получения критики идей и прогресса в работе. Если несколько людей из команды отчаливают и работают над чем-нибудь вместе некоторое время, например пару дней, ничего страшного, но вот когда речь идет о неделях, тогда то, чем они занимаются, следует представить группе как свершившийся факт. Если один человек так поступает, то группа может просто отмахнуться от него. Но если двое или больше – становится сложным откреститься от плохих идей. Секретность и некомпетентность идут рука об руку. Группам, работающим втайне, не постигнуть мудрости.

СОВЕТ: Когда один человек делает что-то в темном углу – это эксперимент. Когда двое или больше делают что-то в темном углу – это тайный заговор.

В случае с ZeroMQ ушло несколько лет на то, чтобы создать по-настоящему открытую и прозрачную атмосферу. До этого главные участники работали тайно, публикуя свою работу только тогда, когда считали, что она готова к общественному обозрению. Но когда они это делали, остальному сообществу было сложно сказать «нет». И зачастую работа была не в тему… да, отличным решением проблемы, но до которой никому нет дела. В конце концов, мы недвусмысленно запретили подобные вещи.

Иронично, что тайна кажется неотъемлемой в некоторых бизнес-моделях. Прибыль часто приходит от игнорирования потребителей. Большинство коммерческих предприятий, даже такие большие сообщества, как Twitter, зависят от строгого разграничения «их» и «нас». Однако цифровое общество лучше всего растет, когда масштаб приоритетней прибылей и когда относится к пренебрежению как к проблеме, требующей решения. Если ваши клиенты не допускаются до ваших внутренних процессов, то вам будет закрыт доступ к пониманию, где в них кроются ошибки.

1.4. Бесплатные участники

Деньги – забавная вещь. Слишком мало – и сообщество будет умирать с голоду (я вернусь к этому позже). Слишком много – начнется разложение. Необходимо понимать, почему каждый из участников вообще занимается этим. Какие у них экономические мотивы? Даже в добровольных сообществах каждый участник преследует свои интересы.

Мы в ZeroMQ изначально начали с малооплачиваемой группы и через два года пришли к добровольному сообществу, прагматично – если не сказать циничнее – умышленно потратив деньги и оказавшись вынужденными уволить разработчиков. Некоторые из них растворились в других компаниях, некоторые вернулись в качестве участников, и проект стал более захватывающим и веселым, чем был раньше. Люди работали над ZeroMQ, потому что им это было нужно для собственных проектов – потратив немного времени на его улучшение, они выигрывали или экономили в разы больше.

Когда вы работаете на кого-то, то будете делать то, что он или она хочет. Когда вы работаете на себя, вы делаете то, что нужно вам. Это огромная разница. Люди с деньгами, но без навыков или вкуса, – шелуха общества. Мы презираем оплачиваемых участников Википедии, платных блогеров и модераторов на Reddit, потому что мы знаем, что выражаемые ими мнения почти по определению ложь. Будет ли блогер, проплаченный Голливудом, критиковать новый летний блокбастер?

Я не имею ничего против наемных сотрудников. Однако если вы нацелены на создание наиболее крупного, наиболее успешного сообщества, то вам нужны участники, который будут стараться по честным, понятным причинам. Если кинорежиссёр приходит на Reddit обсудить фильм – здорово. Если его маркетологи заходят, чтобы потереть критические комментарии, это отвратительно.

СОВЕТ: один бесплатный участник стоит десяти оплачиваемых сотрудников.

Свобода работы с материалами (ремиксабельность)

Группе требуется много договоренностей, чтобы работать сообща. Я называю их «протоколами». Наверно, самый важный из них для творческого сообщества – возможность перерабатывать материал (ремиксабельность). Будь то музыка, искусство, изображения, видео, комментарии, программы или wiki-страницы, встанет следующий вопрос: «А что за авторская лицензия стоит за этим материалом, и как это затронет сообщество?».

Грубо говоря, есть три типа авторских лицензий:

- А) лицензия locked down не позволяет перерабатывать материал. Это старый способ вести дела, и он все еще доминирует в коммерческой деятельности.
- Б) лицензия free to take позволяет одностороннюю переработку. Это доминирующая модель для многих open source сообществ.
- В) лицензия share-alike позволяет двустороннюю переработку. Это преобладающая модель для сообществ бесплатного программного обеспечения, таких как ZeroMQ, и для многих художественных сообществ (хотя это может быть и неписаной договоренностью).

Пользователи предпочитают модель free to take, потому что она позволяет им использовать контент как угодно без обратных обязательств. Представьте себе диджея, который выпускает популярный трек по модели free to take. Потом компания делает ремикс и использует его в рекламе. И этот ремикс будет закрыт для использования. Теперь, диджей не сможет переработать этот ремикс и, возможно, не сможет даже проигрывать этот ремикс.

Все же сообщества работают лучше с третьей моделью, т.к. тогда пользователи становятся участниками. С лицензией share-alike диджей смог бы использовать ремикс, ремикшировать его еще и превратить в дискотечный хит. Знания и идеи текут во всех направлениях, а не вытекают из сообщества в застойное болото. Это мощное течение, и это особенно важно для тех из нас, кто строит сообщества с минимальным бюджетом. Если вы являетесь крупной компанией, вкладывающей кучу денег в сообщество, то модель free to take вам подойдет лучше.

СОВЕТ: Если каждый участник владеет тем, что он привнес в сообщество, а вы используете лицензию share-alike, вам не требуются переуступки авторских прав или возобновление лицензии от участников.

1.5. Четкость протокола

Хорошие протоколы позволяют посетителям участвовать без предварительного одобрения. Они разрешают деструктивные конфликты и превращают их в полезные состязания. То, что анархисты могут присоединяться к умной толпе так же успешно, как и любой другой, объясняется тем, что толпа может разрабатывать свои собственные правила. Обычно эти правила касаются переработки материала, идентичности, рангов и т.д. Не важно, какая у них форма, хорошие правила просты, четки, ясно прописаны и всеми одобрены.

Если вы создаете проект в области программного обеспечения, вы можете взять существующее руководство, например, протокол C4, который мы сделали для ZeroMQ. Или же вы можете начать с минимумом инструкций и добавлять их по мере определения тех проблем, с которыми сталкивается сообщество. К слову, так было и с руководством Википедии. Некоторые правила должны быть установлены с самого начала (например, об авторских правах и участии). Другие могут быть придуманы по мере необходимости (например, процедура разрешения конфликтов). Сложные, бесцельные или не прописанные правила отравляют группу. Они создают пространство для споров, путают людей и повышают стоимость входа в группу или выхода из нее.

СОВЕТ: Пишите аккуратно ваши правила, начиная с лицензии на контент, и оценивайте, насколько они помогают людям. Изменяйте их по мере необходимости.

1.6. Компетентность власти

Без органов власти правила не имеют силы. Основатели сообщества и основные участники являются де-факто их представителями. Если они злоупотребляют своим положением, они теряют участников – и проект умирает либо разветвляется в зависимости от различных правил. Власть должна быть масштабируемая (то есть быть способной охватывать деятельность группы любого размера) и допускать передачу по мере роста и изменения

группы.

Пока мы используем власть для сооружения игровой площадки, многие группы используют власть для контроля своих членов, держа их в группе и заставляя их соответствовать стандартам. Любимый прием в культах – наугад наказывать и вознаграждать людей, чтобы они были сбиты с толку и перестали задавать вопросы администрации.

СОВЕТ: Назначайте самых активных участников на административные посты и побыстрее. У вас есть небольшой промежуток времени, чтобы успеть сделать это, иначе они уйдут в другие проекты.

Вы должны быть частью вашего сообщества, и вы должны соблюдать ваши собственные правила. Если вы замечаете за собой, что нарушаете их или хотите это сделать, значит, они неточны и требуют корректив.

В сообществе ZeroMQ мы сражались из-за вопроса о том, кто мог определять правила, и в конце это привело к торговой марке и доменному имени. Человек или компания, которая владеет именем проекта, является верховной властью и может определять правила. Если они идиоты, то проект умрет.

СОВЕТ: Если вы инвестируете деньги в сообщество, рассмотрите вариант использования торговой марки в США, чтобы иметь возможность предотвращать использование другими людьми похожих имитирующих названий, которые не имеют к вам отношения. Это стоит около 750 долларов.

1.7. Нон-трайбализм

Членство должно быть символом объединения, а не служить удостоверением. Как часто отмечал Мистер Спок, эмоции не логичны. Некоторые группы руководствуются логически обоснованными целями, в других же правят эмоциональные факторы, такие как давление со стороны членов своего круга, стадный инстинкт и даже коллективная истерия. Определяющим моментом, видимо, служат отношения между группой и ее участниками. Мы может выявить это вопросом: участники «исключительно привержены» группе? Под исключительной приверженностью имеется в виду придание большего значения существованию группы, а не ее работе. Подобная приверженность заканчивается конфликтом с другими группами.

СОВЕТ: Держитесь подальше от формальных моделей членства, особенно тех, которые стараются превратить людей в собственность группы. Позволяйте анонимное или не персонализированное участие. Поощряйте людей создавать свои конкурирующие проекты – пространство для экспериментов и для изучения нового.

Группы индустриальной эпохи, словно культы, владеют своими членами. Сотрудник принадлежит компании. Даже идеи, которые пришли вам в голову под душем, – тоже собственность вашего работодателя. А когда группа владеет своими участниками, то мотивирует их страхом, ненавистью, завистью и злостью, подменяя сознательные логичные мотивы. Страх исключения широко используется для подчинения людей одному стандарту: «Делай, что я говорю, или я уволю тебя!».

СОВЕТ: Для определения того, насколько группа похожа на племя, просто начните конкурирующий проект. Если реакция на это отрицательная и эмоциональная, в группе доминирует родоплеменная парадигма. В группе со здоровой атмосферой аплодисментами встретят своих соперников.

1.8. Самоорганизация

Некоторым людям нравится, когда им говорят, что делать. Лучшие участники и команды сами выбирают себе задачи. Успешное общество распознает проблемы и само организуется для их решения. Более того, оно делает это быстрее и лучше, чем любая иерархически управляемая структура. Это значит, что сообщество должно принимать помощь в любой области, без ограничений.

Распределение задач сверху вниз является антипаттерном с присущими ему многими слабостями. Он не дает индивидуумам действовать при обнаружении ими проблемы. Для него характерны феоды, где работа и необходимые ресурсы принадлежат отдельным людям. Он создает длинные коммуникационные цепочки, которые не позволяют реагировать быстро. Ему требуются прослойки менеджеров, просто чтобы соединить тех, кто принимает решения, с теми, кто будет выполнять работу.

СОВЕТ: Пишите правила, чтобы повысить качество работы, подчеркивающие, что каждый может работать над тем, что ему интересно.

В сообществе ZeroMQ мы избавились от назначения задач. Например, мы не принимали запросы о каких-либо особенных функциях. Если кому-то нужна была специальная функция, то он либо посылает нам патч, либо предлагает оплатить добавление изменений, либо он ждет. Это значит, что люди делают только те изменения, которые на самом деле нужно сделать.

СОВЕТ: Сообществам требуется иерархия полномочий. Однако она должна быть подвижной и строго делегированной. То есть выбирайте людей, с которыми вы работаете, и позвольте им выбирать тех, с кем они будут работать. Структура полномочий, словно жидкий цемент, она затвердевает и сковывает движения людей. Любая структура старается себя защитить.

1.9. Толерантность

В разношерстной группе возникают конфликтующие мнения, и здоровая группа эти конфликты охватывает и перерабатывает. Критики, иконоборцы, вандалы, шпионы и тролли держат группу в напряжении. Они могут быть катализатором вовлеченности остальных участников. Википедия процветает благодаря, а не вопреки, тем, кто кликает «Edit» с целью превратить статью в мешанину. Это классический антипаттерн, подавляющий идеи и взгляды меньшинства, используя предпосылку, что они «опасны». К тому же это неизбежно подавляет новые идеи. Логика обычно в том, что слаженность группы важнее ее разнообразия. Потом же получается так, что на ошибки не реагируют, а лишь еще больше обособляются. На самом деле, группа может быть важнее, чем результаты ее деятельности, если она многообразна и открыта новым аргументам. Это трудный урок, который полезен и обществу в целом: нет опасных суждений, есть опасные ответы.

То, как сообщества разбираются с троллями и вандалами, это одно. Разобраться с фундаментальными отличиями мнений – это другое. Ранее я говорил, что конфликтующие друг с другом миссии могут стать проблемой. Лучшее решение, которое я знаю, – это превратить конфликт в состязание. К примеру, браузер Google Chrome стал более легкой, более быстрой альтернативой Firefox, который становился раздутым и медленным. Тогда команда Firefox взялась за дело с умом, и теперь Firefox работает быстрее, чем Chrome.

СОВЕТ: Если есть интересная проблема, сделайте так, чтобы несколько команд соревновались, пытаясь решить ее. Соревнование — очень веселая штука, и может породить лучшие решения, чем монополистический подход. Вы можете даже организовывать соревнования с призами.

1.10. Измеримый успех

Все это хорошо: пытаться обратить конфликт в состязание. Однако вам необходимо обеспечить участников группы информацией о том, как хорошо они справляются. Лучшие инструменты, такие как GitHub, показывают точное число людей, которые наблюдают или отметили проект или начали проект-ответвление (отражены различные уровни интереса и приверженности).

Конечно же, Сеть всегда была озабочена «хитами» и анализом траффика, который показывает популярность сайта или страницы. Это облегчает измерение успеха онлайн-проекта. В старые времена индустриальной эпохи команды получали отзывы о своей работе от начальства. Что превращалось в ужимки перед властью: вас больше наградят за послушность, чем за прилежность. Делать начальство счастливым ради того, чтобы вам повысили зарплату, – не здоровое отношение.

СОВЕТ: Если ваша платформа не поддерживает этого напрямую, найдите возможности информировать ваших участников о том, насколько хорошо развиваются их проекты.

1.11. Высокое награждение

Существует много причин, по которым люди принимают участие в сообществах. Преобладающая мотивация – потребность в восхищении за достигнутый успех. Как индивидуумом, так и в составе команды. Успех относительное явление, поэтому нам требуется метрика, какой-то высокий балл, на который люди будут ориентироваться в своих стремлениях.

В сообществе ZeroMQ мы не придавали большого значения балльной оценке, хотя участники и получают больше любви при большем вкладе в общее дело. Это записывается в их послужной список. Участие в ZeroMQ может помочь при поиске хорошей работы.

Reddit, как многие другие сайты, использует «карму», которая показывает, сколько голосов получил аккаунт за свои публикации и поведение. Работает это довольно неплохо. Некоторые сайты не показывают всю карму, чтобы предотвратить попытки людей обойти систему и получить более высокий балл. Некоторые сайты, такие как StackOverflow, до крайности увлекаются «геймификацией», используя ордена, высокие баллы, достижения и т.д. Мне кажется, это отдает манипуляциями и отвлекает от миссии сообщества. Люди

должны принимать участие, стремясь к успеху проекта, а не к большому количеству игровых баллов.

Социальное обязательство – делать группы разных людей счастливыми – задача, приносящая огромное удовлетворение, и она не загрязняет планету. Индустриальное общество нацелено на материальные награды (выше зарплата, больше дом, лучше машина), увязанные с иерархической структурой. Оно эффективно, потому что все мы любим богатство или у нас комплекс неполноценности; какая бы ни была причина, желание сделать начальство счастливей значит принятие меньших рисков.

СОВЕТ: Когда люди просят вас сделать что-то, а вы не знаете как, тогда объявите публично, что это «невозможно». Или предложите решение настолько нелепое и безнадежное, что настоящие эксперты от возмущения возьмутся за дело.

1.12. Децентрализация

В своей книге Сероуиеки (Surowiecki) объясняет, что катастрофа шаттла «Колумбия» произошла по причине бюрократии в иерархичной структуре управления NASA, из-за которой были проигнорированы мнения обычных инженеров. Если группа децентрализована, ее члены более независимы, они получают большее различных входных данных, и они с самого начала разнообразны.

Если группа географически не разбросана, то она становится однородной, где все члены обладают схожими входными данными и триггерами. Схожесть позволяет меньшинству доминировать над настроем группы и отбрасывать неординарные идеи. Оно позволяет ему буквально запугивать или обманывать большинство, тем самым подчиняя его. Требование о том, чтобы все члены группы сидели в одном офисе, департаменте или здании является старым антипаттерном, который сложно преодолеть. Вот почему все культы такие сплоченные.

СОВЕТ: Вам нужны собрания, чтобы добиться от группы работы? Это знак того, что у вас есть глубокие проблемы в совместной работе. Вы исключаете людей, которые физически не находятся рядом.

Бывает сложно отойти от старой модели совместной работы «обсуждение-действие». И, конечно, вам будет легче, если вы собираете группы с самого начала, а не пытаетесь изменить уже существующие.

1.13. Свободная рабочая среда

Сообществу нужно пространство для роста. В реалиях интернета это обычно сайт или набор сайтов и сопутствующие структуры вроде списков эл. почты, блогов и т.д. Мы видим, что это становится очень дешевым или даже бесплатным способом создания «пространства» в цифровом обществе. Вопрос в том, могут ли индивидуумы создавать свои личные пространства внутри сообщества. Если да, будут ли они приносить больше пользы общему проекту?

Свобода создания структуры раздражает людей, которые считают, что это вносит хаос и

беспорядок. Однако если вы используете обычные структуры (смотрите следующий пункт), ущербу участникам от этого нет никакого. А вот что вредно, так это создание структуры исходя из необоснованного мнения о ее пользе людям. Когда я возглавил ассоциацию FFII в 2005 г., предыдущим президентом было создано несколько сотен списков эл. почты, так он отмечал те проекты, над которыми, по его мнению, люди должны были работать. Это не соответствовало тому, как люди хотели быть организованы, и было очень сложно удалить эти списки и создать новые, которые нам на самом деле были нужны.

Конечно, группы индустриальной эпохи распределяли работу и ресурсы для ее выполнения. Любая новая инфраструктура – такая как сайт, список адресов эл. почты или вики – требует одобрения и решительности. Может даже потребоваться юридическая оценка авторских прав и патентов. Цена высока, поэтому люди неохотно идут на риск. Получается, что не экспериментируя и продолжая работать, они связывают себе руки.

В сообществе ZeroMQ требуется лишь один клик для создания нового проекта. В Википедии вы можете создать новую страницу, просто кликнув на «создать страницу». Оба проекта имеют механизмы защиты от случайного мусора. Википедия проводит довольно агрессивную чистку новых страниц. В ZeroMQ есть специальная процедура для внесения проекта в официальную организацию сообщества.

СОВЕТ: Сделайте создание новых проектов для зарегистрированных пользователей максимально простым. Если проект создается пользователем, то не стоит беспокоиться насчет мусора. Если они находятся в общем пространстве, то вам могут потребоваться инструменты для очистки мусора и заброшенных проектов.

1.14. Стандартная структура

По мере того как сообщество растет, направлять его становится сложнее. Если вы делаете единичный, постоянно развивающийся проект, состоящий из множества отдельных задач, то это становится все сложнее и сложнее со временем. Представьте себе средневековый замок. Эта проблема особенно остро стоит перед большими компаниями, развивающими проект, которые иногда забывают о затраченных средствах.

Запутанность отпугивает людей – очень сложно становится разобраться. Решением является использование стандартных структур, выучив которые раз, вы можете распознавать всегда. Подойдет не любая структура. Нам бывает сложно выучить структуры глубже трех-четырех уровней. Однако мы с радостью исследуем очень широкие системы с тысячами или миллионами блоков, если эти блоки соответствуют отдельным задачам или проектам.

Представьте себе город.

Успешные онлайн-сообщества – это города, а не замки. Википедия состоит из нескольких вики со специфичным языком, многие разбиты на миллионы страниц (проектов), каждая структурирована секциями, обсуждением, историей, примечаниями и т.д. Несколько людей могут работать над одной страницей одновременно или один человек может медленно править или заботиться о дюжине или сотне страниц.

GitHub управляет миллионами программных хранилищ («репозиториями»),

сгруппированными по учетным записям пользователей или организаций, и каждый обладает своей структурой (файлы-исходники, документация и т.д.), что зачастую зависит от языка (Java-репозитории используют один стиль, C-репозитории – другой и т.д.). Один репозиторий может насчитывать несколько участников, люди могут работать с разным количеством репозиториев. Сообщество ZeroMQ является организацией, которая состоит из растущего числа проектов.

СОВЕТ: Спроектируйте ваше сообщество как город поддающихся поиску проектов, где любой может начать новый проект – проекты представляют, наверно, дюжину работ людей, и у всех схожая структура, насколько это возможно. Бизнес тяготеет к замкам, которые неизбежно будут посвящены Важным Персонам, а не проектам, и, конечно же, не основным проблемам бизнеса. Эти организации всегда огромны и нестандартны. И нет никакой возможности разобраться в них, кроме как запоминать каждую их деталь. Но и тогда вы не сможете с легкостью прогуливаться по замку, поэтому мало толку изучать его планировку.

1.15. Плавность обучения

Когда ZeroMQ только начинался, это был лишь один проект с единственной страницей README. Сегодня это сто или больше небольших проектов, каждый из которых имеет свою документацию, сообщество и динамику. Попасть в уже взрослый проект может быть трудно. Как я уже сказал, использование стандартных структур жизненно необходимо. Более того, вам потребуется проследовать по довольно специфической траектории при изучении, от легкого к более сложному, от стадии праздного посетителя до участника-эксперта. Считайте ваше сообщество компьютерной игрой, где сложность уровней возрастает соразмерно с выигрышем. Люди будут играть «в соответствии со своим уровнем». Если вы все делаете правильно, вы привлечете многих. Если неправильно, то эксперты будут скучать на легком уровне, а новичков отпугнет сложность на старте.

СОВЕТ: Используйте классические инструменты обучения – презентации, видео, ответы на часто задаваемые вопросы (FAQ), обучающие материалы – чтобы люди могли начать. Вам будет легче, если вы состоите в сообществе, потому что тогда вы можете посмотреть, какие вопросы чаще всего задают начинающие.

1.16. Позитивность

Иногда есть соблазн агрессивно агитировать людей вступить в сообщество. В конце концов, многим нравятся острые аргументы, особенно когда они уверены в своей правоте. Некоторые группы развиваются за счет своей враждебности и негатива по отношению к другим группам, особенно если еще есть и предыстория. Тон, который вы задаете, будучи основателем, сохранится на долгое время. Если вы продвигаете свое сообщество, нападая на конкурентов, вы привлечете определенно настроенных людей, и такой настрой получит развитие. Рано или поздно негатив обернется внутрь и может оказаться губителен для сообщества.

СОВЕТ: Когда вы говорите о людях, продукции или организациях, будьте вежливы и не теряйте самоконтроль. Когда вы рекламируете продукцию или сообщество, говорите о проблемах, которые вы решили, а не о том, чем вы лучше конкурентов.

На своем опыте знаю, что лучше задать позитивный тон с самого начала. Конкуренты – это благо, т.к. дают на возможность состязаться. Подражатели – тоже хорошо, потому что подтверждают, что рынку вы нужны. Тролли и вандалы – отлично, ведь они дают искренним людям дополнительный шанс доказать свою полезность. И так далее. Кажется, что это трудно, искать позитивную сторону во всем. Однако это всего лишь образ мышления.

СОВЕТ: Добро пожаловать всем, за исключением безнадежных смутьянов. Их немного, тех, которые просто не могут найти себе место в открытом, разнообразном сообществе. Вы можете попросить таких людей уйти, или, если необходимо, забанить их.

Позитивный настрой характеризуется толерантностью и препятствует накалу страстей и возникновению споров. Так легче экспериментировать, делать ошибки и критически относиться к своей работе, а все это вместе взятое позволяет сообществу решать сложные задачи.

1.17. Чувство юмора

Вы когда-нибудь задумывались, почему в человеке заложена потребность шутить, почему люди, которые никогда не смеются, кажутся странными и неприветливыми? Моя теория такова, что мы все используем юмор как способ разрядить ситуацию (что имеет очевидное преимущество для выживания). Люди не побьют шутника – только если шутка старая или плохо рассказана. Если серьезно, то юмор сводит на нет трайбализм и эмоции, и позволяет людям работать вместе, даже если они сильно отличаются друг от друга. Общая шутка может создать сильные связи, потому что она свидетельствует о схожести взглядов. Юмор является неотъемлемой частью общества и уменьшает стресс.

СОВЕТ: Чем более серьезную тему затрагивает ваше сообщение, тем больше вам потребуется юмора. В моей книге о ZeroMQ мной написано много глупой чепухи вперемешку с тяжелым описанием технической части. Многим это понравилось. Если бы не алкоголь, то хмурое выражение лица индустриальной экономики никогда бы не сменялось улыбкой. Она себя очень серьезно воспринимает. Недостаток юмора в организации – явный признак того, что все там фундаментально убого. Хуже всего, что тогда группа становится уязвимой для конфликтов и расколов.

1.18. Минимализм

Гоночные машины делаются быстрее за счет избавления от лишнего веса, а не увеличения мощности. Вы можете сделать ваше сообщество более легким, быстрым и гибким, строго следуя догме минимализма по отношению к вашей работе. Это может смахивать на леность, но часто бывает сложнее не делать что-то веселое, чем ввязаться в это без оглядки.

Общее правило – делайте всегда тот минимум объема работ, которого будет достаточно, чтобы все работало. Остальное будете делать, когда люди начнут использовать вашу работу и жаловаться. Это относится как к вашему посевному проекту, так и к каждому изменению, которое вы вносите. Обратная связь – в большей степени, чем ваше собственное мнение, – лучший указатель того, где следует приложить усилия.

СОВЕТ: Перфекционизм препятствует участию. Публикация на половину оконченной и содержащей баги работы – прекрасный способ привлечь людей к участию. Для больших эго это трудно принять, но изъяны лучше привлекают участников, чем совершенный труд, который привлекает пользователей.

Культура минимализма может и должна распространиться в вашем сообществе. В прошлом мы обычно создавали юридические лица для серьезных проектов, чтобы была возможность владеть авторскими правами, торговыми марками и деньгами. Однако содержание юридических лиц дорого обходится и занимает много времени. Одна налоговая отчетность может стать непосильным бременем.

Одно из моих сообществ, Digistan, было разработано, развито и достигло своего результата (формирование нового поколения узаконенных шаблонов и политических аргументов для открытых стандартов) где-то за шесть месяцев. Все наши протоколы в ZeroMQ основываются на работе Digistan. Open Web Foundation, занимаясь теми же вопросами, потратило два года только на регистрацию юрлица, определение устава и выбор сотрудников.

1.19. Разумное финансирование

Если средств будет недостаточно, сообщество умрет от истощения. Если их будет слишком много, то, как я уже говорил, оно будет разлагаться. Здесь требуется чуткое равновесие. Мы можем мотивировать людей с помощью денег до определенной степени. После этого, только психопаты будут показывать пропорциональную реакцию. В этом и заключается дефект наивной теории капитализма о том, что «чем больше денег, тем лучше». В моем деле самыми вероломными оказывались те, кому я больше всего платил.

Первое, что нужно сделать для сокращения расходов, – это отказаться от идеи с юрлицами, офисами и сотрудниками, если только вы правда в них не нуждаетесь. Они не только съедят любые ваши средства, но они будут препятствовать вашей работе по созданию только онлайн-сообщества.

Второе: инвестируйте время и деньги в сообщество, только если нет другого выхода. Это может относиться к регистрации торговой марки, оплате хостинга или плате за выполнение такой работы, с которой больше никто не справится. И наконец: остерегайтесь людей, которые готовы взять на себе серьезные риски без требования соответствующего вознаграждения, – они склонны перегорать, о чем я расскажу в следующей главе.

СОВЕТ: Каждый раз, собираясь потратить деньги на сообщество, поинтересуйтесь, не может ли кто-нибудь вам помочь с проблемой.

Chapter 2. Sidebars

In the previous chapter, I examined my toolbox for building on-line communities. Now I'll look at few other key ideas that are worth knowing about.

2.1. The Market Curve

The [https://www.google.com/search?q=marketing+curve market curve] is a well-known theory of marketing that is less known in engineering and community building. However it's important to understanding how communities develop over time. In the classic market curve, a new technology, idea, or product enters the market as a wave, starting with ice-breaking enthusiasts and pioneers, then the early adopters, then the mass market, then the late adopters, and finally the skeptics.

Each of these groups has different motivations for coming to a project, joining in, and eventually, leaving. If we take an exciting new technology like ZeroMQ, we can explore this and understand how it works:

- When the project is young and experimental, it attracts pundits and researchers whose business is new stuff, in general. These people need to know why the project is different from what exists, what its goals are, and why it is exciting. They will never use it, nor will they become contributors. They are your evangelists. They often lose interest rapidly.
- When you have a seed product, it attracts pioneers. These are hard-core hackers who want the latest stuff and don't care about documentation, marketing, or tutorials. They're very good at managing the risk of new things. These are your first wave of contributors. Often they are building frameworks for other developers.
- When you have a real, usable product, it attracts early adopters. These are people making real products yet who are good at taking and managing risk. They still don't need much help, though they do expect some guarantee that things won't break randomly. This is the bulk of your community.
- When you are in version two or three, you will start to attract the mass market. These are people who expect stability and reliability. They'll ask questions like, "Do you offer support?" Some of these will become contributors. Mostly, however, they are the target paying customers.
- Finally, when you are in later versions, the laggards and skeptics will finally pick up older versions and try them.

It's more complex than this, as you can have multiple overlapping curves. You need to keep the whole market interested, or you lose valuable sections of your community. Each section sells to the next, so you should aim new versions at the evangelists so they can sell them to the pioneers, and so on.

Once you understand the market curve, you see why it's counterproductive to, for instance, write perfect tutorials for the early versions. You won't get the mass market regardless and it will feel patronizing to the pioneers.

2.2. Эмоциональное выгорание волонтеров

Ранее я подчеркивал ценность добровольной работы как более аккуратной, честной и творческой по сравнению с оплачиваемой работой. Однако здесь нужно сделать важную оговорку. Некоторые инструменты Социальной Архитектуры таят в себе опасность. Поставив людям захватывающую цель, вы можете подтолкнуть их в сторону саморазрушения. Это было главной проблемой в FFII (Foundation for a Free Information Infrastructure), когда я пришел туда, и ее усугублял высокий накал эмоций, характерный для родоплеменной корпоративной культуры организации в то время. Многие ключевые участники были изнурены и эмоционально истощены. Не понаслышке знакомое мне самому состояние.

Исследования эмоционального выгорания, о которых вы можете прочитать в Википедии, на мой взгляд, не соответствуют тому, что происходит в реальной жизни. А реальность всё-таки важнее теории. Я неоднократно наблюдал такую характерную особенность выгорания в добровольных сообществах:

- Оно выражается как глубокое отвращение к конкретному проекту. Мы забрасываем проект подальше, прекращаем отвечать на эл. почту и можем даже покинуть сообщество. Остальные отмечают: «хм, он ведет себя как-то странно… наверно, расстроен или устал».
- Это состояние проектоориентированно. Т.е. мы выгораем по отношению к одним проектам, а с другими все может быть нормально. В иных случаях нас может парализовать даже на несколько месяцев — потом мы опять начнем работать, но уже забросив проект и взявшись за что-то другое.
- Такое случается раз в один-три года, в зависимости от вашего характера и ситуации. Очень упорные, мотивированные личности могут выдержать дольше, но когда сорвутся, будет еще хуже.
- Лечение есть. Это наистраннейший подход я опробовал, когда удалось найти денег и заплатить перегоревшим добровольцам за то, что они делали бесплатно. Они вернулись счастливыми и продолжили заниматься своим делом.
- Недуг также можно предотвратить. Оплачиваемые работники не страдают также от истощения. Конечно, и они могут впасть в депрессию, но обычно их не вырубает внезапно.

Что приводит меня к мысли о том, что дело тут в проблеме оптимального вложения профессиональных усилий.

Люди много вкладывают в свои профессии, идут на большой риск, особенно в молодости в надежде на награду потом. Мы долгое время можем не придавать значения материальному вознаграждению, если мы уверены в том, что следуем верным путем. Например, молодой писатель или музыкант готов терпеть бедность многие годы, если он считает, что слава и богатство ждут его впереди.

И не важно, насколько пожухшая морковка свисает перед нами с палки — она всегда маячит у нас в подсознании. Мы по своей сути экономические животные. Вся жизнь — бухгалтерия. Мы превосходно умеем врать себе, но все же за каждым действием и

решением стоит экономический мотив. Мы инвестируем в проекты, потому что чувствуем, что они будут способствовать нашему успеху, даже если на это уйдут годы. Мы соревнуемся с другими, пытаясь найти ниши, где наши таланты засияют во всей красе.

И вот получается так, что молодой мозг, изо всех сил старающийся инвестировать свои ресурсы в правильные вещи, обнаруживает себя в ситуации, когда снежный ком лжи достигает критической массы. Дорога внезапно заканчивается тупиком. Люди, которые ей следовали, — обманщики и манипуляторы. Миссия — фальшивка. Одобрения со стороны других — эмоциональное манипулирование. Годы усилий идут прахом, и каждая следующая минута становится бессмысленной жертвой.

Подобное выгорание похоже на расплату (like a reckoning). Мы бросаем проект так, будто он вдруг стал ядовитым, словно поняли, что проглотили какой-то протухший кусок. Вот несколько способов снизить вероятность подобного развития событий:

- Мы не можем в одиночестве работать над проектами. Концентрация всей ответственности на одном человеке, который не работает сверх меры, часто приводит к перегоранию.
- Проектам требуется бизнес-план. Надежда получить материальное вознаграждение позволяет мозгу мириться некоторое время с отсутствием вознаграждения.
- Превентивной мерой против выгорания может быть информация. Когда мы объясняем людям, что такое выгорание, они быстрее его распознают и просят о помощи еще до того, как дело заходит слишком далеко.
- Хорошие инструменты и практики позволяют нам работать с меньшим стрессом и с меньшей зависимостью от одного человека.

2.3. Миф об индивидуальном интеллекте

Наверно, вы уже поняли, что я не сторонник одинокой гениальности. По большому счету это объясняется тем, что, несмотря на членство в организации Менса (крупнейшая, старейшая и самая известная организация для людей с высоким IQ), я помню, как совершал на удивление гениальные ошибки. Со временем я стал думать, что любое упоминание об индивидуальном интеллекте является опасно упрощенным мифом.

В нем всегда гениальные личности думают над важными проблемами, и в результате упорного труда они генерируют решения и шлифуют их до совершенства. Иногда они переживают «эврика-моменты», когда им открываются гениально простые ответы на сложные проблемы. Изобретатель, его процесс изобретения, исключительное, драгоценное явление, и остальным лучше не вмешиваться. История полна героями-одиночками. Мы обязаны им нашим современным миром.

Однако если присмотреться внимательнее, то станет понятно, что эта сказка не соответствует фактам. История не показывает одиноких изобретателей. Она рассказывает нам о везении людей, которые украли или присвоили себе право собственности на идеи, над которыми работали многие. Она полна примеров про гениальных людей, которые после удачного попадания тратят десятилетия на бесполезные и безрезультативные поиски. Самые известные крупные изобретатели вроде Томаса Эдисона были хороши в

систематическом поиске, который выполнялся крупными командами. Это как заявить, что Стив Джобс изобрел каждую примочку, сделанную командой «Apple». Этот приятный миф годится для маркетинга, но он далек от истины.

История последних десятилетий, которая лучше зафиксирована и которой сложнее манипулировать, наглядно это демонстрирует. Интернет точно является одной из самых инновационных и быстро развивающихся технологий, о становлении которой имеется большое количество достоверной информации. У этой технологии нет изобретателя. Вместо этого есть огромная масса людей, которые тщательно и успешно решали длинную серию текущих проблем, записывали свои ответы и делали их доступными для всех.

Инновационная природа Интернета обеспечена не маленькой избранной группой Эйнштейнов. Она обеспечена RFC-документами, которые могут быть кем угодно использованы и улучшены, сотнями и тысячами умных, хотя и не уникально умных, людей, программным обеспечением с открытым кодом, который любой может использовать и улучшать. Она происходит из обмена, смешивания и масштабирования сообщества. Она происходит из постоянного увеличения числа хороших решений и избавления от плохих.

Хотя, вот и альтернативная теория инноваций:

1. Есть безграничная область проблем/решений. Словно область равнин и холмов, которые мы пытаемся преодолеть. Решения интересных проблем находятся на вершинах холмов.
2. Область меняется с течением времени в зависимости от внешних обстоятельств. Горы могут превратиться в равнины, а новые горы могут возникнуть там, где их не было, со временем.
3. Мы можем точно воспринимать только те проблемы, которые ближе к нам. У нас нет возможности охватить взглядом все и нам остается полагаться на наши догадки. Наш метафорический ландшафт очень туманен.
4. Мы можем прикинуть, что нам даст и во сколько обойдется задача, оценивая решения. Т.е. мы можем понять, насколько высоко мы находимся.
5. Есть оптимальное решение для любой решаемой проблемы. Так, у любого склона есть вершина.
6. Мы можем достичь это оптимального решения механически, шагнув в примерно правильном направлении и посмотрев, оказались ли мы выше, либо ниже.
7. Наш интеллект может ускорить этот процесс, но не заменить его. Если мы умнее — возможно мы будем шагать быстрее или чуть дальше видеть сквозь туман, и все.

Есть несколько последствий этого:

- Индивидуальная креативность значит меньше, чем сам процесс. Более умные люди могут работать быстрее, но они могут и следовать в неправильном направлении. Быть честными и объективными нам помогает коллективное видение реальности.
- Нам не нужны дорожные карты, если у нас хорошо налажен процесс. Со временем, по мере того, как ценность решений будет расти, будет расти и функциональность.
- Мы не столько изобретаем решения, сколько открываем их. Соболезнования творческим

натурам: это всего лишь обрабатывающий информацию голем, начищающий свое собственное эго и озабоченный поднятием кармы.

- Интеллект — это социальный эффект, хотя он и ощущается как что-то личное. Человек, отрезанный от других, перестает думать. Мы не можем ни определить проблемы, ни оценить их решения без других людей.
- Размер и разнообразие сообщества является ключевым фактором. Более крупные и разнообразные сообщества охватывают больше релевантных задач, решают их более точно и делают это быстрее маленькой группы экспертов.

Поэтому когда мы доверяемся экспертам-одиночкам, они делают классические ошибки. Они фокусируются на идеях, а не на проблемах. Они фокусируются на неправильных проблемах. Они делают неправильные выводы о ценности решаемых проблем. И они не пользуются тем, над чем работают.

2.4. Коллективный Индекс Интеллекта или КИИ (CII)

Я собираюсь предложить инструмент по измерению интеллекта сообщества, другими словами, как точно и эффективно сообщество работает в любой взятый период времени. Он также показывает, насколько приятно будет участвовать в сообществе.

Для его демонстрации я ранжирую несколько сетей, организаций, сайтов и онлайн-сообществ. Это не наука, просто творческая и небрежная прикидка. Как всем известно, 87% статистики изобретается на месте, и 91% людей принимают это без вопросов. Я выбрал следующие жертвы:

1. Википедия
2. Твиттер
3. Реддит
4. Фейсбук
5. Индустрия моды
6. Нигерийский кинематограф, т.н. Нолливуд (Nollywood)
7. Адвокаты как профессия
8. Киноиндустрия Голливуда
9. Сеть The Fox News
10. Военные (в какой-то случайной восточной стране)

Я не буду судить о ценности отдельно взятого сообщества. Это невозможно, и будет обманчиво. Миссия Твиттера — «набрать больше подписчиков» — звучит слабее, чем у Википедии «собираем знания всего мира». Однажды сформированная, умная и гибкая толпа может запросто создавать новые миссии, например «свергнуть диктатора». Онлайн-сообщество, возможно, ценно (для человечества) не благодаря своей продукции, а само по себе. В случае Википедии или ZeroMQ сложно отделить толпу от контента. А в случае Твиттера это очевидно. Контент — явление преходящее и зачастую бесполезное, а толпа —

нет.

Я придумал такую оценочную таблицу:

Критерий	**1.Wk**	**2.Tw**	**3.Rd**	**4.Fb**	**5.Fa**
Четкая миссия	5	3	2	1	2
Свободное участие	5	5	5	5	4
Прозрачность	5	3	5	1	2
Бесплатные участники	5	5	5	5	2
Ремиксабельность	5	5	5	4	4
Четкость протокола	5	5	5	4	4
Компетентность власти	5	4	5	3	4
Нон-трайбализм	4	5	5	5	3
Самоорганизация	5	5	5	5	4
Толерантность	5	5	5	5	4
Измеримый успех	5	5	5	5	5
Высокое награждение	3	5	5	5	4
Децентрализация	5	5	5	5	5
Свободная рабочая среда	5	5	5	5	3
Стандартная структура	4	5	5	5	3
Плавность обучения	5	5	5	4	3
Позитивность	5	5	5	5	5
Чувство юмора	5	5	5	5	2
Минимализм	5	5	4	4	3

Критерий	**1.Wk**	**2.Tw**	**3.Rd**	**4.Fb**	**5.Fa**
Разумное финансирование	5	4	3	3	5
Итоговый счет	**96**	**94**	**94**	**84**	**71**

Критерий	**6.Nw**	**7.Lw**	**8.Hw**	**9.FN**	**10.Ml**
Четкая миссия	1	0	0	0	2
Свободное участие	3	0	1	2	2
Прозрачность	1	0	0	0	0
Бесплатные участники	3	3	2	1	0
Ремиксабельность	3	3	1	1	0
Четкость протокола	3	2	3	1	4
Компетентность власти	3	1	1	0	1
Нон-трайбализм	3	0	2	0	0
Самоорганизация	4	2	2	0	0
Толерантность	3	2	3	0	0
Измеримый успех	5	4	5	5	2
Высокое награждение	3	3	2	1	1
Децентрализация	1	1	1	0	1
Свободная рабочая среда	2	0	0	0	0
Стандартная структура	3	0	1	0	0
Плавность обучения	2	3	3	1	5
Позитивность	3	0	2	0	0

Критерий	6.Nw	7.Lw	8.Hw	9.FN	10.Ml
Чувство юмора	3	0	1	1	0
Минимализм	4	1	1	3	0
Разумное финансирование	3	3	3	2	2
Итоговый счет	**56**	**28**	**34**	**18**	**20**

Если мы можем измерить КИИ сообщества или организации, значит, мы можем улучшить его, уделив внимание аспектам с низкими оценками. В теории это должно сделать организацию умнее, а ее участников счастливее. Конечно, довольно характерно, что военная организация может работать только с низким КИИ. Умная армия, скорее всего, просто разойдется по домам и переключится на Reddit.

2.5. Как захватить/защитить open-source проект

На «Ars Technica» есть интересная статья о том, как Google понемногу закрывает Android. Это классическая игра Capture the Flag, которая ведется против open-source сообщества. Я собираюсь объяснить, как этот захват работает, и как его предотвратить.

Почему Capture the Flag?

Как говорит «Ars Technica»: «Легко отдать что-нибудь, когда ты на последнем месте с нулевой долей рынка, как это было с Android в начале. Когда же ты на первом месте, немного сложнее быть таким открытым и доброжелательным».

Android, если уж честно, вероятно, самая крупная инвестиция Google. Вы можете поспорить о том, имеют ли они право превращать открытую систему в закрытую, и вы будете правы. Однако это то же самое, что спорить о том, имеет ли право центральный банк печатать слишком много денежных знаков и создавать девальвацию. Конечно, на это он уполномочен. Но в то же время у этого существует цена, которую заплатят другие люди. Вопрос не в правомерности, а в приемлемости той цены, которую заплатит общество. А если она неприемлема, тогда как это предотвратить?

Android, как и любая система с открытым кодом, проданная рынку на этой основе, является общественной собственностью. Когда кто-либо приватизирует ее, он увеличивает свои прибыли, как печатающий деньги центральный банк, за счет остальных. Делая форк таких приложений Android, как поиск, календарь, музыка, и создавая улучшенные их версии, Google соперничает с другими компаниями, использующими Android на своих устройствах.

Вопрос о захвате, о том, как это происходит и как это предотвратить, особенно важен, если вы не Google, т.е. если вы пользователь или участник open-source-проекта. В Android много патчей других фирм, таких как LG, Samsung и прочие. По мере того как Google превращает операционную систему в свой личный огород, эти патчи начинают использоваться против

тех же самых людей, которые их сделали.

Я уверен, что Google совершает огромную ошибку, меняя правила игры подобным образом, просто потому что это будет потворствовать конкурентам Android. Однако я не об этом. Я просто заинтересован в усвоении любых уроков, которые помогут мне с моей работой и моими проектами.

Отмечу две вещи:

- из чистого интереса я не буду участвовать в open-source-проекте, который не предоставит мне, участнику, гарантий того, что мои патчи и изменения не станут принадлежать кому-либо еще и не будут использоваться против меня же.
- из соображений этики я никогда не создам open-source-проект, который не будет обеспечивать подобные гарантии своим участникам.

Сценарий использования

Я постараюсь выразиться недвусмысленно о сценарии использования. Речь идет об Android: одна компания начинает open-source-проект, используя его как «товар-приманку», намереваясь проникнуть на рынок, и просит поддержки у других. Это классическая стратегия, которая может быть очень успешной. Однако это точно не то же, что студенческий проект-исследование или мусор вроде «давайте сделаем систему расчетов по заработной плате open-source» или «пятеро из нас собрались в гараже и решили сделать новый фреймворк».

Здесь есть частичное совпадение, и я думаю, полученные выводы можно применять более широко (и я точно применяю их систематически), опять же, мой сценарий использования – «open-source для прорыва на рынок».

Важно знать, что успех использования open-source-проекта для прорыва на рынок зависит от сообщества, которое за него берется. Любой рынок зависит от поведения нескольких влиятельных игроков, доминирующих на рынке, а усилия большинства остальных игроков несущественны. Суть в том, чтобы обещать этой удрученной бессилием толпе выход, убедить их инвестировать во что-то новое и открытое, которое потенциально может изменить правила игры.

Большинство open-source проектов провальные (серьезно, идите, почитайте о каком-нибудь случайном проекте на GitHub и увидите, сколько из них адекватных), и даже успешные в очень скромном значении этого слова, незначительны сами по себе. Пока нет серьезного изменения власти, проект может оставаться потенциальным прорывом на рынке очень долгое время. Он может выглядеть очень стабильным и счастливым. Что ж, легко быть дружелюбным, когда на кону не стоят деньги.

Если и когда проект становится успешным, правила игры меняются, умные парни, которые запустили прорывной проект, стараются сорвать спелый фрукт и забрать его себе. И вот только тогда становится интересно.

Поле с равными условиями игры не под «запретом»

Есть несколько способов захватить open-source проект, включая торговые марки и патенты. Я рассмотрю только авторские права, потому что это наиболее частый случай. Ключевыми соглашениями, которыми регулируются авторские права на open-source проект, являются а) лицензия и б) политика участия.

Частым заблуждением является мысль о том, что open-source проект не может быть захвачен. Это совершенно не верно. Грубо говоря, есть три типа соглашений об авторских правах:

1. «закрытая» лицензия, которая не позволяет повторно обрабатывать материал, классическое авторское право плюс некоторые ограничительные лицензии;
2. лицензия «free to take», которая позволяет одностороннюю обработку материала, например Apache/BSD/MIT;
3. лицензия «share-alike», которая позволяет двустороннюю обработку материала, например GPL, LGPL и cc-by-sa.

Представьте себе ди-джея, который выпускает популярный бит по модели «free to take». Ведущий музыкальный лейбл делает из бита ремикс и выпускает его. Тот становится хитом. И теперь эта новая версия закрыта. Ди-джей не может ремиксить эту новую работу, и, возможно, не может даже проигрывать ремикс. Конечно, он может взять свою старую версию и улучшить ее, однако деньги будет приносить коммерческая версия.

Надеюсь, вы понимаете, к чему я клоню. Даже лучший индивидуальный талант не сможет конкурировать на равных с крупной фирмой с ее маркетинговым и денежным ресурсами. Единственный способ гарантировать равные условия игры в войне за контроль над развитием — двустороннее соглашение об обработке материала. Двустороннее значит, что касается обеих сторон.

Когда люди называют эту гарантию ограничением, остается только вздыхать по этому поводу. Это как называть замок в моей машине «ограничением», потому что он останавливает остальных от присвоения моей машины. Назвать защиту от воров «ограничением» это…. ну, по меньшей мере, неумение анализировать. Когда правила работают для обеих сторон, это не ограничение, ОК?!

Как происходит захват?

Давайте еще раз определимся с целью. Необходимо предотвратить захват open-source проекта кем-то с большими деньгами и властью, кто нацелился собрать урожай с проекта для своей личной выгоды, за счет сообщества, которое помогало развивать или создало проект. Мне все равно, насколько «правомерен» будет этот захват, я просто объясняю, как его предотвратить.

Лицензия и политика участия являются двумя половинками одной головоломки.

Кто владеет авторскими правами? Они «сконцентрированы» у основателей проекта или они разделены между всеми участниками? Это жизненно важный вопрос. Если они сконцентрированы, то это тривиальная задача по покупке авторских прав, разветвлению

проекта, изменению лицензии в одностороннем порядке, — и можно двигаться в закрытом направлении. Однако если права распределены, т.е. многие люди владеют работой, совместно владеют, то вам нужно одобрение всех (не большинства, а 100% единодушие) для изменения лицензии. А это логистически невозможно.

Кстати, если бы вы только знали, сколько людей мне предлагали деньги за коммерческую лицензию на ZeroMQ, вы были бы поражены (очень много). Предложение простое: я продаю им лицензию «non-LGPL», они платят мне хорошие деньги и делают свои версии ZeroMQ. Если бы я специально не позаботился о невозможности этого варианта давным-давно, то я бы был очень богатым. И теперь мириться с бедностью мне помогает осознание того, что ZeroMQ переживет меня.

Давайте еще раз пройдемся по проблеме с предложением коммерческих лицензий для совместной работы. Представьте себе клуб, который приглашает ди-джеев и микширует их биты. Потом клуб оставляет за собой авторские права и продает их звукозаписывающей компании, которая делает свой альбом ремиксов, которые первоначальные ди-джеи уже не могут проигрывать бесплатно. Поэтому да, я считаю dual-GPL / коммерческое лицензирование порочными практиками.

Никто не будет платить за коммерческую лицензию проекта «free to take», потому что они могут просто взять код и использовать его. В некоем смысле я считаю, что это уже неправильно, т.к. нарушает равенство правил игры для всех. Ведь очевидно, что крупная компания выиграет от этого больше, чем маленькие команды. Опять же представьте себе независимого ди-джея, противостоящего звукозаписывающим лейблам со всеми их маркетинговыми и медийными связями и доходами от концертов.

Теперь перейдем к шагу по захвату номер два: найм разработчиков.

«Но код все еще свободен!», — говорят люди. Конечно. Возвращаемся к лейбл vs ди-джей. Пусть лейбл нанимает только одного ди-джея, ключевого сотрудника и использует его, чтобы протолкнуть коммерческий микс альбома. Куда публика тогда пойдет?

Вам не нужно нанимать всех участников в сообществе, чтобы захватить его. В любом случайно взятом проекте будет два-три топовых участника и огромная масса младших. Наймите двух топов и вы можете забрать проект куда угодно. Если результаты могут повторно обрабатываться (ремиксабельны), то это путешествие будет полностью справедливо по отношению к тем, кто участвовал в проекте раньше. А если не ремиксабельно, то все остальные участники обнаружат, что их инвестиции используются против них.

Предотвращая захват

Я знаю только одну модель, которая предотвращает захват open-source проекта в области ПО:

1. Лицензия семейства «GPL» (или MPLv2, которая работает схожим образом).
2. Распределенные авторские права

Именно так я строю open-source проекты с самого начала, и это требование к любому

сообществу, к которому я присоединяюсь. Ваше право делать деньги не включает мое право использовать мою работу как конкурентное преимущество, если только это не взаимовыгодно.

2.6. Legal primer: Trademarks

Trademarks. What are they, do you need them, and how much do they cost? These are questions that often crop up when we build open source projects. Trademarks can be key to protecting a project from bad actors. Yet there is little advice on line. So here is my guide to using trademarks in open source. This is practical advice, IANAL, and certainly not your lawyer.

A Background to Trademarks

Definitions first. A trademark is a name, phrase, logo, or even a specific color (the "mark") that you're using for business ("trade"). The simple fact of using a mark for some period of time establishes the trademark. However as with all property, the devil lies in enforcement. The question is, always, if you go before a judge with a complaint, what standards of evidence will the judge expect and demand?

No matter the case, criminal or civil, it always comes down to convincing one or more humans. If you ever go to court, keep this in mind. The facts of a case, as each party knows them, are irrelevant. How those facts are documented and presented is all that matters.

Let's back up a little and ask why courts even care about protecting businesses' trademarks. First, it's to protect consumers from misleading sales tactics. Just selling junk isn't an offense as such, except when there are legal minimum standards for health and safety. However selling junk that claims to be a more expensive, well-known brand is an offense. So secondly, trademarks let businesses distinguish themselves and stop unfair competition.

So the judge in a trademark violation case will ask, "Was the intent to deceive the consumer? Would a reasonable consumer be deceived?" And then the judge will ask, "Who owned the trademark, and can they prove it?" Even though the simple act using a mark creates it (under so-called Common Law), that can be hard to establish.

For instance, business A creates a chain of restaurants. Business B opens a competing chain using the same colors and similar name. B is clearly hijacking A's investment in branding, stealing goodwill. Yet when A takes B to court, B produces a document showing their restaurant plans, a full year before A started. How does the judge know who is the liar?

In clear cut cases, you can convince a judge that a copycat is deceiving consumers and stealing your goodwill. Yet the risk of losing such a case is high. It's also costly for courts to deal with such cases. Judges may simply refuse to hear them.

Hence most countries provide a way to register your marks, for a fee. Registration gives you a dated document that establishes your claim to the mark. The trademark office does the job of searching for prior marks in the same area. Before it grants you the registration, it publishes your claim and gives others a chance to dispute it. So after a search, and if there are no disputes, a judge will take the trademark registration as solid evidence.

It is not that simple. A competitor can still claim that their Common Law mark outweighs your registered trademark. They can argue that the registration does not represent real goodwill. This is often understood as, "if you don't enforce your mark, you will lose it," which is inaccurate. As trademark holder you're not expected to police the world. However you are expected to be truthful in court when the judge asks you, "are you using your mark, and suffering real damage due to the unfair competition?"

Finally, courts consider trademarks to apply per segment of the market. So you can have XYZ Car Co, and XYZ Clothing Co, with no confusion to the market. When you register a mark you'll need to explain what "classes" you're using it in. You'll probably want international class 9, which is anything that beeps.

Where and How to Register

If you are large enough to need to register in multiple countries then you are large enough to have trademark lawyers. For the rest of us, it's a bit like buying a domain name. Sure, there are hundreds of domain extensions. Yet we still want a dot-com for our main business.

So it is with trademarks. If you decide to register a mark, do it in the US (via the USPTO) first. That's cheap, and simple. Then over time you can register in the EU (via the OHIM), if you find your project is worth it.

The cost for a US registration is around USD 1500, depending on what lawyer you use. You can find trademark attorneys on line. They'll ask you for details of the mark, proof that it's being used, name and address of the registrant, and credit card details. The process takes about six months. After nine years (and before ten years have passed) you can renew the mark.

Getting a US registration will speed up registration in other countries, if you decide to apply for that later. The risk, and it's a small one, is that a troll will register your trademark in some other country, effectively excluding you from doing business under that name, there.

Before you register, however, ask yourself "what is the chance someone would rip off my name and logo?" If it's low, don't bother. If it's high, then ask "what is the chance a cheat would take this to court?" If that is still low, then don't bother either.

Instead of registering a mark you can raise its visibility. This means being explicit on your website and other materials. "X, Y, and Z are trademarks of MyCorp." This scares off potential cheats, improves your case, if you do try to defend the mark in court, and makes it easier to get registration if and when you need it.

How to Enforce your Trademark

Registered or not, you enforce your mark by telling the other party, in writing, "stop now, or else." If they do not stop, you repeat the warning, with initial claims of damages. If they do not stop, you add on more damages and when you have a solid file, you take it to court.

The vast majority of people will back-off at once. The trouble is when you face someone who's well aware of trademark law, has cheap legal resources, and enjoys time in court.

If you are facing such a firm, and you did not register your mark, you should probably fold your

hand, and change your name. The risks are high that you would lose, and have high legal fees and possibly damages to pay. Judges don't always get it right.

If you did register your mark, then you should push ahead and claim damages. You will win, if you stick to the basic rules (you're still using the mark, the damages are real.) Do I need to say, any court case will have to happen in the country of registration? Judges in Belgium won't accept paper from the USPTO.

Trademarks For Open Source Projects

The common misconception about open source is that because the code is free, it does represents no property nor value. The opposite is true: successful projects represent considerable value, owned by many. How does a trademark represent and protect that value?

It comes down to authenticity and reputation. If you download a package calling itself "XYZ v2.0", then you may have expectations. It is compatible; it works; it has no trojans or advertising; it is from the same people as "XYZ v1.0".

If a successful project does not register its name, then anyone can fork it, repackage it, and use the same name. Imagine competing, incompatible versions of "Linux."

When a person or a business registers the name as a trademark, those incompatible forks may still exist. However they may not use the mark. If they try to do that, it's damages time.

I've had this happen at least once in my own projects, and the trademark was the tool I used to stop the incompatible forks and punish the perpetrators. Trademark law is clear enough that saying "trademark violation" will stop 99% of cheats dead still. Producing a registration filing number stops 99% of the remainder.

In a serious project like ZeroMQ you'll end up with three or four marks you want to register, over a period of five to ten years. Register only when it's worth it. That is, to protect real trademarks that you would be willing to defend in court. Consider that in the worst case you might have to spend ten or twenty times the cost of registration, to defend your mark. You might get that back, or you might not.

I hope this small brief has helped you understand trademarks, and how to use them (or not) in your open source projects. And, if someone claims you're infringing on their trademark, how to defend yourself. (Hint: ask them for a registration number.)

Chapter 3. Сообщество ZeroMQ

> Можем ли мы целенаправленно строить сообщества?

Меня иногда спрашивают, что такого особенного в ZeroMQ. На это я всегда отвечаю, что ZeroMQ — возможно лучший ответ, который у нас есть на злободневный вопрос

«Как создавать распределенные программные средства, которые требуются от нас 21 век?».

Но, помимо этого, ZeroMQ выделяется благодаря своему сообществу. Что и отличает волков от овец.

Есть три основных open source паттерна. Во-первых, когда крупная фирма выбрасывает на рынок код, чтобы расправится с конкурентами. Это модель Apache Foundation. Во-вторых, когда крошечные команды и маленькие компании строят свою мечту. Это наиболее распространенная open source модель, которая может быть наиболее коммерчески успешна. И наконец, агрессивные и разнообразные сообщества, всей толпой пробирающиеся сквозь дебри проблем. Это модель Linux, и вот к ней мы и стремимся в ZeroMQ.

Сложно переоценить мощь и упорство работающего open source сообщества. Наверно, не существует лучшего способа создания программного обеспечения в долгосрочной перспективе. Сообщество не только занимается решением самых релевантных проблем, но и делает это оптимально, аккуратно, наблюдая за результатами годами, десятилетиями, пока они сохраняют значение, после чего спокойно оставляет их.

Чтобы получить максимум пользы от ZeroMQ, вам необходимо понимать сообщество. В какой-то момент вам захочется предложить релиз, патч или аддон. У вас может возникнуть желание попросить кого-нибудь о помощи. Вы можете захотеть принять коммерческое участие в ZeroMQ, и когда я предупрежу вас, что сообщество намного, намного важнее, чем компания, поддерживающая его продукцию, хотя я и являюсь исполнительным директором компании, это должно о многом сказать вам.

В этом разделе я рассмотрю наше сообщество с разных сторон и в конце подробно расскажу о нашем договоре о сотрудничестве, который мы называем «C4». Вам пригодится эта информация для вашей собственной работы. Кроме того, мы успешно адаптировали метод ZeroMQ C4 для проектов с закрытым кодом.

3.1. Архитектура сообщества ZeroMQ

Вы знаете, что ZeroMQ является проектом с лицензией LGPL (примечание автора: мы склоняемся к Mozilla Public License v2, которая действует также, но является более простой). На самом деле это набор проектов, построенных вокруг корневой библиотеки libzmq. Я представляю эти проекты как расширяющуюся вселенную:

- В центре находится libzmq, как корневая библиотека ZeroMQ. Она написана на C, с низкоуровневым C-API. Код довольно своенравен, в основном потому, что он сильно оптимизирован, а также потому, что написан на C, на языке, который сам по себе

довольно коварный и дрянной. Большую часть оригинального кода написал Марти Сустрик (Martin Sustrik). Сегодня десятки людей занимаются поддержкой его различных частей.

- У libzmq есть около 50 биндингов. Это индивидуальные проекты, которые создают высокоуровневые API для ZeroMQ, или, по крайней мере, отображают низкоуровневый API на других языках. Эти привязки различаются по качеству, от проб пера до шедевров. Наиболее впечатляющей из них является PyZMQ, которая была одной из первых проектов сообщества ZeroMQ. Если вы разрабатываете биндинг, вам просто необходимо изучить PyZMQ — вдохновившись, вы сделаете ваш код и ваше сообщество такими же крутыми.
- У многих языков есть многочисленные биндинги (Erlang, Ruby, C#, например), написанные разными людьми в разное время, с использованием разных подходов. Мы никаким образом это не регулируем. Нет «официальных» биндингов. Используя тот или другой, вы голосуете за него, участвуете в его развитии или игнорируете его.
- Есть несколько новых вариантов использования libzmq, начиная с JeroMQ, полный Java перевод библиотеки, который сейчас является базой для NetMQ, C# стек. Эти стеки предлагают похожие или идентичные API и используют тот же протокол (ZMTP), что и libzmq.
- На базе биндингов существуют тысячи проектов, которые используют ZeroMQ или построены на нем. Большинство из них не являются частью официального сообщества, лишь некоторые из них, например Zyre и Malamute.

Libzmq, большинство биндингов и некоторые из внешних проектов присутствуют в «организации» сообщества ZeroMQ на GitHub. Эта организация «управляется» группой наиболее младших авторов биндингов. Там мало чем нужно управлять, т.к. оно почти самоуправляемо и в настоящее время конфликтов не наблюдается.

iMatix, моя компания, играет специфическую роль в сообществе. Мы владеем торговыми марками и следим за каждой, чтобы быть уверенными в том, что если вы скачаете пакет с названием «ZeroMQ», вы можете быть доверять его содержимому. Было несколько попыток несанкционированного использования имени, видимо пираты думали, что «свободное программное обеспечение» значит ничейное и беззащитное. Прочитав эту главу, вы увидите, как серьезно мы воспринимаем работу, проделанную над нашим программным обеспечением (а под «нами» я имею в виду сообщество, а не компанию). iMatix поддерживает сообщество, следя за функционированием всего, что называется «ZeroMQ». Мы также вкладываем деньги и время в программное обеспечение и его развертывание.

Мы не занимаемся благотворительностью. ZeroMQ является коммерческим проектом, и он довольно прибыльным. Прибыль делится на большое количество людей, участвовавших в его развитии. Все обстоит именно так, незамысловато: потратьте время на то, чтобы стать экспертом в ZeroMQ или надстройте что-то полезное поверх ZeroMQ, и вы выиграете от роста как индивидуум, команда или компания. iMatix получает те же выгоды, что и все остальные в сообществе. Это взаимовыгодные отношения для всех, кроме наших конкурентов, которые понимают, что с этой угрозой им не справится и не миновать ее. ZeroMQ будет доминировать в будущем мире широко распределенного программного обеспечения.

Моя компания не просто охраняет сообщество — мы также создавали сообщество. Мы занимались этим целенаправленно: в оригинальном техническом описании ZeroMQ от 2007 г. указано, что было два проекта. Один был технический, нацеленный на создание лучшей системы доставки сообщений. Второй заключался в построении сообщества, которое могло бы привести программное обеспечение к успеху. Программное обеспечение умирает, но сообщество живет.

3.2. Как создавать по-настоящему большие архитектуры

Существуют два способа сделать по-настоящему масштабное программное обеспечение, как уже говорилось (по крайней мере, теми, кто читает это предложение вслух).

Первый способ

— бросить огромные объемы денежных средств и проблем на усмотрение умных людей и надеяться, что результатом будет не крест на вашей карьере. Если вы очень везучи, и создаете продукт, основываясь на огромном опыте, и вы смогли удержать команды в сборе, и не стремитесь к техническому совершенству, и впредь удача от вас не отвернется, это сработает.

Но азартные игры на чужие сотни миллионов подходят не всем. Для всех прочих, кто хочет создать масштабное программное обеспечение, есть

Второй способ

— open source, точнее, свободное программное обеспечение. И если вы спрашиваете, какая связь между лицензией программного обеспечения и его масштабом, то это правильный вопрос.

Гениальный и прозорливый Эбен Моглен (Eben Moglen) однажды сказал, что лицензия свободного программного обеспечения словно контракт, на основе которого строится сообщество. Когда я первый раз это услышал, около десяти лет назад, у меня появилась идея — можем ли мы целенаправленно строить сообщества?

Полученный десять лет спустя ответ — «да», более того, теперь это почти наука. Я говорю «почти», потому что у нас до сих пор нет достаточного количества доказательств того, что люди делают это целенаправленно, нет документации и уверенности, что этот процесс возможно точно воспроизвести. Именно это я и пытаюсь исправить с помощью Социальной Архитектуры. ZeroMQ появился после Wikidot, после Digital Standards Organization (Digistan) и после Foundation for a Free Information Infrastructure (FFII, негосударственная организация, которая борется против патентов на программное обеспечение). Все это появилось после множества менее успешных сообществ, вроде Xitami и Libero. Вывод, который я сделал из долгой работы с различными проектами, заключается в следующем: если вы хотите создать по-настоящему масштабное и долговечное программное обеспечение, стремитесь к построению сообщества свободного программного обеспечения.

3.3. Психология архитектуры программного обеспечения

> Один из принципов Социальной Архитектуры заключается в том, что способ нашей организации важнее того, кем мы являемся.

Диркжан Октман (Dirkjan Ochtman) обратил мое внимание на определение архитектуры программного обеспечения в Википедии: «совокупность структур, требуемых для понимания системы, которая объединяет элементы программного обеспечения, связи между ними и их принадлежность». Для меня эта бессодержательная и цикличная болтовня служит хорошим примером того, как унизительно мало мы знаем о том, что на самом деле важно при создании масштабной архитектуры программного обеспечения.

Архитектура — это искусство и наука создания крупных искусственных структур, используемых человеком. Если я что и понял и успешно применял на протяжении тридцати лет при создании все более крупных систем программного обеспечения, так это то, что **программное обеспечение — это все о людях**. Крупные структуры сами по себе бессмысленны. Важно то, как они функционируют для использования их людьми. А в программном обеспечении, человеческое начинается с программистов, которые делают его.

Основные проблемы в архитектуре программного обеспечения кроются в человеческой психологии, а не в технологиях. Наша психология по-разному может влиять на нашу работу. Я могу привести примеры того, как группа людей словно становится глупее по мере того, как она расширяется, или когда им приходится работать, будучи разделенными огромным расстоянием. Значит ли это, что чем меньше команда, тем она эффективней? Как же тогда такое крупное глобальное сообщество как ZeroMQ умудряется успешно работать?

Сообщество ZeroMQ возникло не случайно. Его конструкция была целенаправленно разработана — мой вклад в те ранние дни, когда на чердаке в Братиславе появился код. Разработка основывалась на моем научном питомце, «Социальной Архитектуре», которую Википедия определяет как «сознательная разработка среды, которая поощряет проявление определенных паттернов социального поведения в целях достижения какой-либо цели или целей». Мое определение более конкретно: «процесс или продукт планирования, разработки и создания онлайн сообщества».

Один из принципов Социальной Архитектуры заключается в том, что **способ нашей организации важнее того, кем мы являемся**. Одна и та же группа, организованная по-другому, может выдать совсем другие результаты. Мы как пиры в сети ZeroMQ, и наши коммуникационные паттерны существенно влияют на наше поведение. Обычные люди при налаженных связях могут превзойти группу экспертов, использующих плохие паттерны поведения. Если вы являетесь разработчиком крупного ZeroMQ приложения, вам придется помогать другим находить правильные паттерны совместной работы. Сделайте это хорошо, и ваш проект ожидает успех. Сделайте это плохо, и ваш проект провалится.

И так, вот мой короткий список психологических элементов Социальной Архитектуры:

- **Глупость:** наша ментальная шина имеет пределы, поэтому в какой-то момент мы все можем тупить. Архитектура должна быть простой для понимания. Это правило номер

один: простота важнее функциональности, всегда. Если вы не можете вникнуть в структуру серым холодным утром понедельника до того, как выпить кофе, значит, она слишком сложна.

- **Эгоистичность:** мы действуем только из эгоистических побуждений, поэтому архитектура должна создавать пространство и возможность для эгоистичных поступков, от которых выиграют все. Эгоистичность зачастую является косвенной и неявной. Например, я могу потратить несколько часов, объясняя что-то кому-то, потому что это может пригодиться мне самому позже.
- **Лень:** мы делаем множество предположений, которые потом оказываются неверными. Мы радуемся, когда можем с минимальными усилиями получить результат или проверить предположение быстро, поэтому архитектура должна предусматривать такую возможность. Т.е. она должна быть простой.
- **Зависть:** мы завидуем другим, а это значит, что мы преодолеем нашу глупость и лень, лишь бы доказать, что они не правы, и что мы можем их превзойти. Поэтому архитектура должна предусмотреть пространство для публичных соревнований, с четкими и понятными всем правилами.
- **Страх:** мы не желаем идти на риск, если есть шанс, что мы можем выглядеть глупо. Страх поражения является главной причиной того, что люди становятся конформистами и следуют за большинством, даже если оно ошибается. Архитектура должна позаботиться о том, чтобы люди могли просто и недорого проводить эксперименты скрытно, достигать успеха без наказания в случае неудачи.
- **Взаимодействие:** мы приложим усилия, потратим деньги, но накажем за жульничество и принудим к исполнению справедливых правил. Архитектура должна устанавливать строгие правила, которые будут указывать, как людям работать вместе, а не на то, над чем им работать.
- **Конформизм:** мы с радостью поддаемся конформизму, из-за страха или лени, т.е. если паттерны поведения хорошие, понятно изложены и задокументированы, и обязательны, мы естественным образом каждый раз будем выбирать правильный вариант поведения.
- **Гордость:** мы очень беспокоимся за наш социальный статус, и мы будем усердно трудиться, только чтобы не выглядеть глупыми или некомпетентными на публике. Архитектура должна обеспечить, чтобы каждая часть нашей работы была подписана, чтобы мы бессонными ночами ворочались в кровати и переживали о том, что другие скажут о нашей работе.
- **Жадность:** мы крайне хозяйственные животные (см. эгоистичность), поэтому архитектура должна экономически стимулировать нас тратить ресурсы на достижение результата. Пусть это будет шлифовка наших профессиональных навыков, или буквально получение денег за некие навыки или компоненты. Неважно какой, но экономический стимул обязан присутствовать. Думайте об архитектуре, как о рынке, а не как об инженерной конструкции.

Эти стратегии годятся как для крупных, так и для маленьких организаций или команд.

3.4. Важность контрактов

> Проект, у которого хорошо написан контракт, определяющий условия его завершения, развалится с намного меньшей вероятностью.

Давайте обсудим спорный, но важный вопрос о том, какую лицензию выбрать. Я бы выделил «BSD» вместе с MIT, X11, BSD, Apache и прочими похожими лицензиями, и «GPL» с GPLv3, LGPLv3 и AGPLv3. Главным отличием является распространение прав на любые версии форков, что защищает любую организацию от захвата программного обеспечения, и тем самым делая его «свободным».

Технически лицензия на программное обеспечение не является контрактом, ведь вы ничего не подписываете. Но в широком смысле удобно считать ее именно контрактом, т.к. она подразумевает обязательства всех сторон и позволяет принуждать к их исполнению в суде, в соответствии с авторским правом.

Вы можете спросить, зачем нам вообще нужны контракты при работе с open source? Ведь главное доброжелательность, бескорыстная совместная работа людей. Вы уверены, что принцип «лучше меньше да лучше» всегда здесь уместен? Не значит ли, что больше правил — меньше свободы? Нам на самом деле нужны адвокаты, чтобы рассказывать, как нам работать вместе? Кажется циничным и даже контрпродуктивным насаждать ограничения и правила в счастливом open source, в сообществе свободного программного обеспечения.

Но настоящая натура человека далеко не так прекрасна. Мы на самом деле ни ангелы и ни дьяволы, а просто своекорыстные победители, последние звенья единой цепи победителей длинной в миллиард лет. В бизнесе, сердечных делах или при совместной работе мы либо боремся и спорим, либо оставляем их.

Посмотрите на это с другой стороны: у совместной работы есть два крайних исхода. Либо неудача, несущественная и бесполезная, в случае которой любой нормальный человек спокойно уйдет. Либо успех, существенный и ценный, в случае которого мы начнем борьбу за власть, контроль и, часто, за деньги.

Хорошо написанный контракт как раз защищает те ценные отношения от конфликта. Супружеские отношения с меньшей вероятностью окончатся разводом, если его условия были четко оговорены заранее. Деловое предприятие, в котором стороны оговорили решение различных типичных конфликтов, например, когда одна сторона присваивает клиентов, либо материальные ценности другой стороны, с намного меньшей вероятностью закончится раздором.

Аналогично проект по программному обеспечению, у которого хорошо написан контракт, определяющий условия его завершения, с намного меньшей вероятностью развалится.

Альтернативной кажется вариант с поглощением проекта более крупной организацией, которая страхом потери обеспечения и бренда сможет сплотить команду. По своему опыту знаю, что у этого есть своя цена, и часто это заканчивается получением преимуществ более богатыми участниками (которые могут позволить себе иногда огромные расходы).

В open source проекте или проекте по свободному программному обеспечению, распад

обычно принимает форму форка, когда сообщество разделяется на две или более группы, у каждой из которой есть свое видение будущего. Во время медового месяца, который может растянуться на годы, проекту не страшен разрыв. Но вот когда проект начинает стоить денег, или когда основные его авторы эмоционально выгорают, добросовестность и благородство улетучиваются.

Поэтому при обсуждении лицензий на программное обеспечение, когда речь идет о вашем коде или используемом вами коде, немного цинизма не повредит. Не задавайтесь вопросом: «какая лицензия привлечет больше последователей?», т.к. ответ зависит от формулировки миссии и процесса участия. Спросите себя: «если проект окончится боем и разделится на три части, какая лицензия спасет нас?». Или: «если всю команду подкупит враждебная фирма с целью присвоения себе кода, какая лицензия убережет нас?».

Долгое выживание требует быть стойкими в тяжелое время, но позволяет наслаждаться хорошими временами.

Когда BSD-проекты ветвятся, они не могут с легкостью слиться опять. На самом деле, когда возникает односторонний форк BSD-проекта, планомерно происходит следующее: BSD-код становится частью коммерческого проекта, вот что происходит. Когда же случается форк GPL-проекта, его слияние — обычное дело.

Приведу уместную здесь историю о GPL. Хотя сообщества программистов, работающих с открытым кодом, уже были широко распространены в 1980-х годах, они все еще использовали простые лицензии, которые работали до того момента, пока проект не начинал привлекать настоящие деньги. В то время был солидный текстовый редактор Emacs, изначально построенном на Lisp Ричардом Столлманом. Другой программист Джеймс Гослинг (который потом явил нам Java) переписал Emacs на C с помощью сообщников, предполагая, что он будет открытым. Сталлман взял этот код за основу для своей C версии. Гослинг позже продал код компании, которая взяла и заблокировала возможность для кого бы то ни было распространение конкурирующего продукта. Столлман посчитал эту продажу совместной работы крайне не этичным поступком и начал развивать многоразовую лицензию, которая бы защитила сообщества от подобного.

В итоге это вылилось в Универсальную общественную лицензию GNU (GNU General Public License), которая использовала традиционное авторское право для защиты возможности повторной переработки материла (ремиксабельности). Это был элегантный прием, который переняли и в других сферах, например, Creative Commons для фотографий и музыки. В 2007 г. вышла в свет версия 3 лицензии, которая была ответом на запоздалые атаки Microsoft и прочих. Она превратилась в длинный и сложный документ, но корпоративные специалисты по авторскому праву хорошо знакомы с ним, и на моей памяти очень мало компаний осмеливаются использовать программное обеспечении библиотеки под лицензией GPL, при условии, что границы обозначены четко.

Таким образом, хороший контракт — а я считаю, что современная GPL идеальна для программного обеспечения — позволяет программистам работать вместе без предварительных соглашений, организаций или убеждений в порядочности и доброжелательности. Сотрудничать становится дешевле, а конфликты оборачиваются здоровой конкуренцией. GPL не просто определяет, что будет с форком, — она поощряет форки как инструмент для экспериментирования и обучения. Где-то с «более либеральной»

лицензией форк может погубить проект, но GPL-проекты развиваются благодаря форкам, потому что успешные эксперименты могут быть обратно включены, согласно контракту, в исходный продукт.

Да, есть много процветающих BSD-проектов и много мертвых GPL-проектов. Обобщать всегда плохо. Судьба проекта зависит от многих причин. Однако в спортивных соревнованиях стоит использовать любые преимущества.

Другой важной чертой противостояния BSD и GPL является «утечка» — так я называю ее потому, что она напоминает мне процесс наполнения водой емкости, на дне которой есть отверстие, небольшое, но существенное для результата.

3.5. Выпей меня

Вот вам история. Она произошла со старшим шурином двоюродного брата друга моего коллеги по работе. Его звали, и все еще зовут, Патрик.

Патрик был специалистом в области информатики с кандидатской степенью в области сетевых топологий. Он потратил два года и свои сбережения на создание нового продукта и выбрал лицензию BSD, т.к. верил, что она принесет ему больше признания. Он работал у себя на чердаке, ущемляя себя во всем, и с гордостью опубликовал работу. Люди аплодировали, ведь работа была просто фантастическая, его эл. почта загудела активностью, патчами и счастливой болтовней. Многие компании рассказывали ему, как много миллионов они сэкономили, используя его работу. Некоторые даже ему заплатили за консультации и обучение. Его приглашали выступать на одну конференцию за другой, хоть бейджики со своим именем собирай. Он начал свой маленький бизнес, нанял друга на работу, стал мечтать о заоблачных вершинах.

Но однажды кто-то показал ему новый проект, под лицензией GPL, который представлял собой форк его работы с некоторыми улучшениями. Он был раздражен, расстроен и все спрашивал, как — друзья по открытому коду! — как они могли подобным бесстыжим образом украсть его код. Тогда было много долгих рассуждений о том, законно ли выпускать его BSD-код под лицензией GPL. Оказалось, что да. Он пытался игнорировать новый проект, но вскоре понял, что выходящие к нему новые патчи уже нельзя слить с его собственной работой!

Дальше хуже: GPL-проект стал набирать популярность, и некоторые основные последователи Патрика начали делать сначала небольшие, а потом все более солидные патчи к нему. И опять он не мог использовать эти дополнения, и тогда он почувствовал себя покинутым. Патрик впал в депрессию, его подружка ушла от него к валютному брокеру, которого, что забавно, зовут Патрис, и он перестал работать над проектом вообще. Он чувствовал себя преданным и до слез жалким. Он уволил своего друга, который воспринял это тяжело и потом всегда очень не лестно о нем отзывался («closet banjo player»). В итоге Патрик устроился на работу на должность проектного менеджера в облачной компании и к сорока годам совсем прекратил программировать ради удовольствия.

Бедный Патрик. Мне его почти стало жаль. Когда я спросил его: «Почему ты не выбрал GPL?», — он ответил: «Потому что ограничивающая вирусная лицензия». «Пусть у тебя и есть докторская степень и пусть ты старший шурин двоюродного брата друга моего коллеги

по работе, но ты идиот, и Моника правильно сделала, что бросила тебя. Ты опубликовал свою работу, предлагая людям украсть твой код, а когда люди сделали именно это, ты расстроился. Что еще хуже, ты вел себя лицемерно, ведь пока они делали это тайно, ты был счастлив, но когда они открыто заявили об этом, ты почувствовал себя, видите ли, покинутым».

Наблюдать за тем, как твою работу захватила более хитрая команда и использует ее против тебя — пытка, так зачем допускать такую возможность? Любой проприетарный проект, который использует BSD-код, захватывает его. Публичный GPL-форк, может показаться оскорбительным, но так вы точно не подставитесь.

BSD — словно лакомство. Я буквально (на самом деле метафорически) слышу шепот «выпей меня», таким тихим голоском, которым, бывает, говорит с вами бутылка лучшего пива в мире — а это, без сомнения, Orval, сваренное древним и почти исчезнувшим орденом молчаливых бельгийских монахов Les Gars Labas Qui Fabrique l'Orval. Лицензия BSD, как и его близкий клон MIT/X11, была специально разработана университетом (Калифорнийским университетом в Беркли) чтобы без корыстолюбивых побуждений выдать работу или усилия. Это был способ протолкнуть субсидируемые разработки по цене ниже себестоимости, ценовой демпинг с целью выхода на рынок. BSD — отличное стратегическое решение, но подходит только крупным, хорошо финансируемым институтам, которые могут позволить себе использовать Первый способ. Лицензия Apache — та же BSD, только в костюме.

Для нас, капитанов малого бизнеса, которые пересчитывают свои средства как последние пули, утечка работы или усилий не приемлема. Здорово было бы перекроить рынок, но мы не можем позволить себе субсидировать наших конкурентов. Сетевой стек BSD привел к появлению Windows в интернете. Мы не можем позволить себе битвы с теми, с кем мы по природе своей должны быть союзниками. Мы не можем позволить себе ошибки фундаментального бизнеса, потому что в итоге нам придется увольнять людей.

Все сводится к поведенческой экономике и теории игр. Тип лицензии, которую мы выбираем, влияет на экономику тех, кто использует нашу работу. В индустрии программного обеспечения есть друзья, враги и пища. BSD выставляет нас в глазах других обедом. Закрытый код — врагами (вам нравится платить людям за программы?). Однако GPL, за исключением Патрика, — союзниками. Любой форк ZeroMQ является лицензионно совместимым с ZeroMQ, до того момента, когда мы поощряем форки в качестве ценных инструментов для экспериментирования. Да, кажется непривычным наблюдать, как кто-то забирает у тебя игрушку и возится с ней, но — вы можете в любой момент взять ее обратно.

3.6. Процесс

Если вы до сих пор соглашались со мной — отлично! Теперь я объясню сам процесс построения open source сообщества. Вот как мы построили или вырастили или чутко ввели сообщество ZeroMQ в мир.

Ваша цель как лидера сообщества — мотивировать людей добраться туда и исследовать, убедить их, что это безопасно для них и для окружающих, награждать их в случае успешных открытий и гарантировать им, что своим знанием они могут поделиться с другими (не потому, что мы просим их, и не потому, что они щедрые, а потому, что таков

Закон).

Это повторяющийся процесс. Вы делаете маленький продукт, за свой счет, но на виду у всех. Потом вы строите маленькое сообщество вокруг продукта. Если у вас маленький, но настоящий хит, тогда сообщество поможет разработать и построить следующую версию, и станет больше.

А потом это сообщество создаст следующую версию и т.д. Очевидно, что при этом вы остаетесь частью сообщества, возможно даже самым главным его участником, но чем больше контроля вы хотите над материальными результатами, тем меньше людей захотят участвовать. Запланируйте свою отставку до того, как кто-то решит, что вы их следующая проблема.

3.7. Безумство, красота и простота

Вам нужна такая цель, которая будет достаточно безумной и простой, чтобы вытащить людей из кровати утром. Ваше сообщество должно привлекать лучших людей, а это требует чего-то особенного. В случае с ZeroMQ мы говорили, что мы собираемся создать «Быстрейшую. Передачу сообщений. Всегда», и это пример хорошего мотиватора. Если бы мы сказали, что мы собираемся сделать «изящный транспортный уровень, который соединит все движущиеся элементы вашего предприятия дешево и гибко», мы бы провалились.

Также ваша работа должна быть прекрасной, полезной здесь и сейчас и привлекать внимание. Ваши участники — пользователи, которые хотят узнать чуть больше, чем они знают сейчас. Сделайте ее простой, элегантной и брутально чистой. Люди должны испытывать эмоции от использования ваших трудов. Они должны чувствовать что-то, и если вы аккуратно решили хотя бы одну большую проблему, которую они до этого даже не осознавали, маленькой частью души они будут с вами.

Ваш труд должен быть простым для понимания, использования и присоединения. Слишком многие проекты обременены препонами для присоединения к ним: поставьте себя на место другого человека и увидьте причины, по которым он пришел к вам на сайт, думая «хм, интересный проект, но…», и потом ушел. Вы хотите, чтобы они остались и попробовали, хотя бы раз. Используйте GitHub и поставьте там трекер задач.

Если вы правильно все это сделаете, ваше сообщество будет умным, но что более важно, оно будет интеллектуально и географически разнообразно. Это на самом деле важно. Группа схоже мыслящих экспертов не сможет хорошо исследовать ландшафт проблемы. Они имеют тенденцию допускать большие ошибки. Разнообразие всегда превалирует над образованностью.

3.8. Незнакомец, позвольте представить вам Незнакомца

Как часто двое людей должны согласовывать свои действия в случае совместной работы? В большинстве организаций, очень часто. Но вы можете свести к нулю эту необходимость, и тогда люди смогут работать, даже не встретившись ни разу лично, не приняв участие в

телеконференции, в деловой поездке, не обсудив Роли и Обязанности в окружении неприлично большой кучи бутылок дешевого корейского рисового вина.

Вам потребуются хорошо написанные правила, разработанные кем-нибудь циничным, вроде меня, чтобы призвать незнакомцев к взаимовыгодному сотрудничеству вместо того, чтобы конфликтовать. GPL будет хорошим стартом. GitHub и с его стратегией «форк-слияние» будут хорошим продолжением. А потом вам потребуется что-то вроде нашей книги правил C4 для контроля того, как на самом деле осуществляется работа.

C4, а я использую ее теперь для каждого нового open source проекта, содержит детальные и проверенные ответы на большинство типичных ошибок, которые совершают люди: например, такой грех, как работа офлайн в укромном месте с другими «потому что это быстрее». Прозрачность имеет ключевое значения для обретения доверия, без чего в свою очередь не будет масштаба. Пусть каждое изменение будет на виду так же, как и весь процесс, и тогда вы сможете полностью доверять результатам.

Другим смертным грехом, в который впадают многие open source разработчики, является мнение о том, что они выше остальных. «Я основал этот проект, к тому же мой уровень интеллекта выше, чем у других». Это не только не скромно и грубо, и часто не верно, это еще плохо для дела. Правила должны распространяться на всех одинаково, без различий. Вы — часть сообщества. Ваша работа, как основателя проекта, заключается не в том, чтобы навязать ваше видение продукта остальным, а в том, чтобы установить хорошие, честные и соблюдаемые правила.

3.9. Неограниченная собственность

Одним из самых прискорбных вымыслов индустрии знаний заключается в том, что идеи являются собственностью. Эту средневековую чушь следует похоронить вслед за рабством, однако она до сих пор приносит слишком много денег слишком многим влиятельным людям.

Идеи дешевы. А вот что является собственностью, так это та тяжелая работа, которые мы делаем, создавая рынок. «Как потопал, так и полопал» — это правильная модель для вдохновения людей на трудную работу. Будь то моральный авторитет в проекте, деньги за консультации, продажа торговой марки богатой и крупной компании: если вы сделали это, вы этим владеете. Но на самом деле ваш главный актив, который определяет ваш потенциал — «посещаемость», участники в вашем проекте.

Для этого потребуется неограниченное количество свободного пространства. К счастью, GitHub решил эту проблему за нас, так что на смертном одре я буду ему благодарен (в жизни слишком много вещей, за что я благодарен, и все здесь не перечислить, т.к. у нас есть только сто страниц или около того, но это одна из таких вещей).

Вы не сможете масштабировать единственный проект со многими владельцами так, как вы могли бы масштабировать несколько небольших проектов, у каждого из которых меньше собственников. Когда мы принимаем форки, человек сможет стать «владельцем», один раз кликнув. И тогда ему нужно лишь убедить остальных присоединиться, продемонстрировав им свою уникальную ценность.

Поэтому в ZeroMQ мы стремились облегчить процесс написания биндингов поверх корневой библиотеки, а сами перестали пытаться их делать. Это дало возможность другим заняться этим, стать их владельцами и поставить себе это в заслугу.

3.10. Забота и поддержка

Я бы хотел, чтобы сообщество было полностью самоуправляемо, и, возможно, когда-нибудь так и будет, но пока это не так. ZeroMQ близко к этому, но по моему опыту сообществу требуется четыре вещи:

Во-первых, просто потому, что большинство людей слишком милые, нам требуется некое символическое лидерство или владельцы, которые будут выступать конечными арбитрами в случае возникновения конфликта. Обычно это основатели сообщества. Я видел, как с этим управляется самоизбранная группа «старших», но старики слишком любят поболтать. Я видел, как сообщества раскалываются, сталкиваясь с вопросом «кто главный?», и создают юридические лица с советом директоров, который только усугубляет споры о контроле. Может так получается, т.к. кажется, что есть, что делить. Но одним из настоящих преимуществ свободного программного обеспечения является его ремиксабельность, поэтому вместо того, чтобы драться за пирог, просто отщипните «вилкой» кусочек.

Во-вторых, сообществам требуются правила жизни, и еще юрист, способный эти правила сформулировать и записать их. Правила критически важны — будучи хорошо составленными, они исключают трения. А неправильно составленные, или игнорируемые, приведут к раздорам и сложностям, которые отпугнут большую часть, оставив спорящую группу во главе горящего дома. Я сам пробовал создать универсальные правила для ZeroMQ и предыдущих сообществ, поэтому, наверно, нам не так уж и нужны юристы.

В-третьих, сообществам нужна некоторая финансовая поддержка. Эти острые рифы потопили не один корабль. Если вы держите сообщество на сухом пайке, оно будет более креативным, но ключевые участники будут эмоционально выгорать. Если вы вольете в него слишком много денег, то привлечете профессионалов, которые никогда не скажут «нет», и сообщество потеряет свое разнообразие и креативность. Если вы создадите общий фонд на раздачу, то люди будут бороться (и яростно) за него. В ZeroMQ мы (iMatix) тратили наши деньги и время на маркетинг и продвижение (вроде этой книги), а также на базовые вещи, например, на исправление багов, релизы и сайты.

И последнее, продажи и коммерческое посредничество также важны. Естественно, есть рыночные отношения между специалистами-участниками и потребителями, но и у тех и у других не очень получается общаться между собой. Потребители считают, что поддержка должна быть бесплатна или стоить очень дешево, ведь программное обеспечение свободное. Участники же слишком стесняются просить достойную плату за свою работу. Это затрудняет рыночные отношения. Все большая часть моей работы и прибыли моей компании обеспечивается деятельностью по соединению пользователей ZeroMQ, которым требуется помощь, с экспертами сообщества, способных ее оказать, таким образом, чтобы обе стороны были довольны результатами.

Я видел загибающиеся сообщества гениальных людей с благородными целями из-за того, что их основатели делали некоторые или все из этих четырех вещей неправильно. Основная проблема заключается в том, что ни одна компания, человек или группа не

может идеально руководить сообществом постоянно. То, что сегодня работает, завтра может не сработать, к тому же структура со временем становится более ригидной, а не гибкой.

Лучшим решением, к которому я пришел, является сочетание двух пунктов. Первый — это GPL, т.к. она обеспечивает возможность повторной обработки материала (ремиксабельность). Не важно, насколько плохим будет руководство, не важно, насколько упорно оно будет стараться приватизировать и захватить работу сообщества — если она под лицензией GPL, то работа просто уйдет и найдет себе руководителей получше. Прежде, чем вы скажете «любой open-source предлагает тоже самое», подумайте. Я могу покончить с проектом с лицензией BSD, наняв ключевых участников и прекратив выпуск новых патчей. Но, даже имея миллиард долларов, я не могу убить проект с лицензией GPL. Второй пункт — отношение к руководству с позиции философии анархизма, которое проявляется в том, что мы выбираем руководство, оно нами не владеет.

Chapter 4. Протокол для коллаборации C4

> «Это эссенция тридцатилетнего опыты разработки программного обеспечения.»

Когда мы говорим о ZeroMQ, мы иногда имеем в виду `libzmq` — основную библиотеку. В начале 2012 года мы синтезировали процесс `libzmq` в формальный и многоразовый протокол для совместной работы, который мы назвали “Контрактом на разработку коллективного кода” или C4. Вы можете рассматривать это как слой над лицензией (например, MPLv2). Это наши правила, и я объясню причины возникновения каждого из них.

C4 — это эволюция модели GitHub Fork + Pull. Вы можете подумать, что я поклонник git и GitHub. И это точно: эти два инструмента оказали положительное влияние на нашу работу в последние годы, особенно когда речь идет о создании сообщества.

4.1. Язык

> Ключевые слова «ДОЛЖЕН», «НЕ ДОЛЖЕН», «ТРЕБУЕТСЯ», «ДОЛЖЕН», «НЕ ДОЛЖЕН», «СЛЕДУЕТ», «НЕ СЛЕДУЕТ», «РЕКОМЕНДУЕТСЯ», «МОЖЕТ» и «ДОПОЛНИТЕЛЬНО» в этом документе следует интерпретировать так, как описано в RFC 2119.

Начиная с RFC 2119, в тексте про C4 четко указано, что он намерен выступать в качестве протокола, а не как случайно написанный набор рекомендаций. Протокол — это договор между сторонами, который определяет права и обязанности каждой стороны. Они могут быть знакомы в сети, или могут быть незнакомцами, работающими в одном проекте.

Я думаю, что C4 — это первый пример того, как кто-то пытался кодифицировать нормативы сообщества как формальную и многоразовую спецификацию протокола. Раньше наши правила были распространены на нескольких страницах вики и были во многом специфичны для `libzmq`. Но опыт учит нас, что чем более формальные, точные и многоразовые правила, тем легче становится незнакомым между собой людям сотрудничать. А меньшее количество разногласий означает более масштабируемое сообщество. Во время C4 у нас также было некоторое несогласие в проекте`libzmq`над тем, какой процесс мы использовали. Не все чувствовали себя связанными одними и теми же правилами. Скажем так, некоторые люди считали, что у них особый статус, и это создавало конфликт с остальной частью сообщества. Таким образом, спецификация сделала вещи понятнее.

Использовать C4 легко: просто разместите свой проект на GitHub, заставьте другого человека присоединиться и откройте функцию pull request. В своем README поместите ссылку на C4, и все. Мы сделали это в целом ряде проектов, и, похоже, это работает. Я не раз испытывал приятное удивление, применяя эти правила к своей собственной работе, например в проекте CZMQ. Никто из нас не бесподобен настолько, что мог бы работать без остальных.

4.2. Цели

> Спецификация C4 предназначена для обеспечения оптимальной неоднократной модели сотрудничества для проектов с открытым исходным кодом.

Вкратце причина создания C4 заключалась в том, чтобы положить конец разногласиям в процессе сотрудничества в `libzmq`. Инакомыслящие ушли в другие места. Сообщество ZeroMQ плавно и легко расцветало, как я и предсказывал. Большинство людей были удивлены этому, но и удовлетворены. Не было никакой реальной критики C4, кроме ее ветвящейся политики, о которой я расскажу позже, поскольку она заслуживает отдельного обсуждения.

Есть причина, по которой я рассматриваю историю создания здесь: как основатель сообщества, вы просите людей инвестировать в вашу собственность, товарный знак и брендинг. В свою очередь, как мы и поступаем в случае с ZeroMQ, вы можете использовать этот брендинг, чтобы установить планку качества. Когда вы загружаете продукт маркированный «ZeroMQ», вы знаете, что он был выпущен по определенным стандартам. Это основное правило качества: запишите свой процесс; иначе вы не сможете его улучшить. Наши процессы не идеальны, они никогда и не станут таковыми. Но любой недостаток в них может быть исправлен и протестирован.

Поэтому очень важно сделать C4 пригодным для многократного использования. Чтобы узнать больше о наилучшем возможном процессе, нам нужно собрать результаты максимально широкого спектра проектов.

> У всего этого есть следующие определенные цели: расширение сообщества, формируемого вокруг проекта, уменьшение порога вхождения для новых участников и создание масштабируемой модели партнерства с позитивной обратной связью.

Целью номер один является размер и жизнеспособность сообщества — не техническое качество, не прибыль, не производительность, не доля рынка. Цель — просто количество людей, которые вносят свой вклад в проект. Наука здесь проста: чем больше сообщество, тем точнее результаты.

> Уменьшение зависимости от ключевых лиц путем распределения требуемых наборов навыков по разным специалистам, чтобы на любую область приходился высокий уровень компетенции.

Возможно, худшей проблемой, с которой мы столкнулись в`libzmq`, была зависимость от людей, которые могли понять код, управлять ветками GitHub и делать чистые релизы — все в одно и то же время. Это похоже на поиск спортсменов, которые могут бегать марафоны и спринты, плавать, а также поднимать вес. Мы, люди, можем быть очень хороши, соблюдая специализацию. Просить нас быть действительно хорошими в двух противоречивых вещах не лучшая идея: это резко сократит число кандидатов, что плохо для любого проекта. У нас

была эта проблема в `libzmq`в 2009 году или около того, и она была решена путем разделения роли Мейнтейнера на две: один человек делает патчи, а другой выпускает релизы.

> Обеспечение более быстрой и точной разработки проекта путем развития процесса принятия решений.

Это теория — не полностью доказанная, но и не сфальсифицированная. Чем больше сообщество и число людей, которые могут участвовать в дискуссиях, не опасаясь быть подвергнутыми критике или увольнению, тем быстрее и точнее разрабатывается программное обеспечение. Скорость здесь довольно субъективна. Быстрое движение в неправильном направлении не просто бесполезно, оно сильно наносит ущерб проекту (а мы испытали много чего в `libzmq`, прежде чем перешли на C4).

> Поддержание жизненных циклов версий проекта от экспериментальной до стабильной, путем проведения безопасных экспериментов, быстрого реагирования на неполадки и изолированием стабильного кода.

Это занимательное влияние на процесс: git-ветка мастер обычно всегда идеально стабильна. Это связано с размером изменений и временем ожидания, т.е. периодом от написания кем-то кода до его полноценного использования кем-то еще. И все же нормальный процесс обучения проектировки обычно повторяется в проектах, пока не становится стабильным и незыблемым.

> Уменьшение внутренней сложности репозиториев, что приводит к облегчению для участия контрибьюторов и уменьшению объема ошибок.

Любопытное наблюдение: люди, которые преуспевают в сложных ситуациях, любят повышать градус запутанности, потому что тогда они сохраняют свою высокую ценность. Это эффект Кобры (загуглите это). Git сделал ветви легкими и оставил нас со слишком распространенным синдромом «git прост, если вы понимаете, что ветвь git — это просто сложенное пятимерное лептонное пространство, которое имеет оторванную историю без промежуточного кеша». Разработчики не должны чувствовать себя глупыми из-за своих инструментов. Я видел слишком много высококлассных разработчиков, запутавшихся в структурах репозитория и не принимающих общепринятую мудрость о ветвях git. Мы скоро вернемся, чтобы разобраться с git-ветвями, дорогой читатель.

> Обеспечение статуса коллективной собственности проекту, что увеличивает финансовую мотивацию участников и снижает риск плагиата со стороны враждебных организаций.

В конечном счете, все мы — экономические существа, и ощущение, что «мы владеем этим, и наша работа никогда не может быть использована против нас», значительно облегчает людям жизнь, чтобы инвестировать в проект с открытым исходным кодом, такой как

ZeroMQ. И это не может быть просто чувство, это должно быть реальностью. Существует целый ряд аспектов для создания коллективной собственности, мы рассмотрим их один за другим, когда мы пройдемся по C4.

4.3. Основные положения

> Проект ДОЛЖЕН использовать распределенную систему управления версиями git.

У Git есть свои недостатки. Его API-интерфейс командной строки ужасно непоследовательный, и у него сложная, неряшливая внутренняя модель, которой он тычет вам в лицо по любому поводу. Но, несмотря на то, что он делает все возможное, чтобы заставить своих пользователей чувствовать себя глупо, git делает свою работу действительно очень хорошо. Более прагматично, я обнаружил, что если вы держитесь подальше от определенных областей (ветвей!), люди быстро учатся git и не совершают много ошибок. Это подходит для меня.

> Проект ДОЛЖЕН быть размещен на github.com или его эквиваленте, называемом здесь «Платформа».

Я уверен, что однажды какая-нибудь крупная фирма купит GitHub и сломает ее, и на ее место встанет другая платформа. До сих пор Github обслуживает почти идеальный набор минимальных, быстрых, простых инструментов. Я направил туда сотни людей, и все они до сих пор там, словно мухи, застрявшие в блюдце с медом.

> Проект ДОЛЖЕН использовать систему управления проектами.

Мы допустили ошибку в`libzmq`, перейдя на Jira, потому что мы тогда еще не научились правильно использовать трекер GitHub. Jira — отличный пример того, как превратить что-то полезное в запутанный беспорядок, потому что бизнес зависит от продажи большего количества «функций». Но даже не критикуя Jira, сохранение трекера задач на той же платформе означает, что на один пользовательский интерфейс, который придется учить, станет меньше, одним логином станет меньше, появится плавная интеграция между проектами и патчами.

> Проект ДОЛЖЕН иметь четко документированные рекомендации по стилю кода.

Это плагин протокола: вставьте здесь правила стиля кода. Если вы не документируете стиль кода, который вы используете, у вас нет оснований, кроме предубеждений, чтобы отклонить патчи.

> «Участник (Contributor)» — это человек, который хочет предоставить патч, являющийся набором коммитов, которые решают четко определенные проблемы. «Мейнтейнер (Maintainer)» — это человек, который объединяет патчи в проекте. Мейнтейнеры не являются разработчиками; их работа заключается в соблюдении процесса разработки.

Теперь мы переходим к определениям сторон и разделению ролей, которые избавили нас от пагубной структурной зависимости от редких людей. Это хорошо работает в`libzmq`, но, как вы увидите, это зависит от остальной части процесса. C4 — не скатерть-самобранка, вам понадобится весь процесс (или что-то очень похожее), чтобы все не рассыпалось на части.

> Участники НЕ ДОЛЖНЫ иметь возможность коммитить в репозиторий, если они не являются также Мейнтейнерами. Мейнтейнеры ДОЛЖНЫ иметь возможность коммитить в репозиторий.

Чего мы хотели избежать, так это того, чтобы люди проталкивали свои изменения непосредственно в мастер-ветку. Это был самый большой источник проблем в`libzmq`исторически: большие массы сырого кода, на стабилизацию которых потребовались бы месяцы или годы. В конечном итоге мы следовали другим проектам ZeroMQ, таким как PyZMQ, с использованием запросов на загрузку. Мы пошли дальше и указали, что все изменения должны идти по тому же пути. Никаких исключений для «особых людей».

> Каждый, без различия или дискриминации, ДОЛЖЕН иметь равное право на возможность стать Участником в соответствии с условиями этого контракта.

Мы должны были указать это прямо. Раньше было так: сторонники`libzmq`отказывались от патчей просто потому, что им это не нравилось. Теперь это может показаться разумным для автора библиотеки (хотя`libzmq`не был написан одним человеком), но давайте вспомним о нашей цели создания работы, которая принадлежит как можно большему количеству людей. Говорить «Мне не нравится ваш патч, поэтому я собираюсь его отклонить», это эквивалентно высказыванию: «Я утверждаю, что владею этим, и я думаю, что я лучше тебя, и я тебе не доверяю». Это токсичные сообщения для тех, кто думает стать вашими соинвесторами.

Я думаю, что эта борьба между индивидуальным опытом и коллективным разумом разыгрывается и в других областях. Она создала Википедию, и до сих пор продолжает это делать, уже спустя десятилетие после того, как превзошла все, что могла бы сделать небольшая группа экспертов. По мне так мы делаем программное обеспечение, медленно синтезируя самые точные знания, так же, как мы делаем статьи в Википедии.

4.4. Лицензирование и собственность

> Проект ДОЛЖЕН использовать такую же лицензию, как MPLv2, или вариант GPLv3 (GPL, LGPL, AGPL).

Я уже объяснил, как полная ремиксабельность (возможность повторной работы с материалом) создает лучший масштаб, и почему MPLv2 или GPL и их варианты кажутся оптимальным контрактом на ремиксабельное программное обеспечение. Если вы крупный бизнес, нацеленный на то, чтобы сбрасывать код на рынке, вам не нужен C4, но тогда вам и нет дела до сообщества.

> Все вклады в исходный код проекта («патчи») ДОЛЖНЫ использовать ту же лицензию, что и для проекта.

Это устраняет необходимость в какой-либо конкретной лицензии или соглашении об участии в разработке патчей. Вы делаете форк MPLv2 или GPL кода, публикуете свою переделанную версию на GitHub, и вы или кто-либо еще можете отправить это как исправление к исходному коду. BSD этого не допускает. Любая работа, содержащая BSD-код, может также содержать нелицензионный проприетарный код, поэтому вам нужно разрешение от автора кода, прежде чем вы сможете его переделывать.

> Все патчи принадлежат их авторам. НЕ ДОЛЖЕН присутствовать никакой процесс присвоения авторских прав.

Здесь мы подходим к основной причине того, что люди уверены в своем вкладе в ZeroMQ: логически невозможно купить авторские права на создание конкурента с закрытым исходным кодом для ZeroMQ. iMatix тоже не может этого сделать. И чем больше людей посылают патчи, тем сложнее это становится. ZeroMQ не просто свободен и открыт сегодня — эта его особенность позволит ему оставаться таким всегда. Обратите внимание, что это не относится ко всем проектам MPLv2 / GPL, многие из которых по-прежнему требуют возврата авторских прав своим мейнтейнерам.

> Каждый Участник ДОЛЖЕН быть ответственным за идентификацию себя в Списке участников проекта.

Другими словами, мейнтейнеры не являются карма-бухгалтерами. Любой, кто хочет добиться одобрения, должен сам заявить об этом.

4.5. Требования к патчу

В этом разделе мы определяем обязательства Участника: в частности, что представляет собой «годный» патч, чтобы у Мейнтенеров были правила, в соответствии с которыми они могут принимать решения о принятии или отклонении патча.

> Мейнтейнеры и Участники ДОЛЖНЫ иметь учетную запись на Платформе и ДОЛЖНЫ использовать свое настоящее имя или известный псевдоним.

В худшем случае, когда кто-то разместил вредный код (запатентованный или принадлежащий кому-то другому), мы должны уметь отслеживать, кто это сделал и когда, чтобы мы могли удалить код. Указывать настоящие имя или известный псевдоним — это теоретическая стратегия по снижению риска появления фиктивных патчей. Мы не знаем, работает ли это, потому что у нас еще не было проблем с этим.

> Патч ДОЛЖЕН представлять собой минимальное решение конкретной идентифицированной и согласованной проблемы.

Это реализация принципа Ориентированной на простоту разработки, про который я расскажу в этой главе позже. Одна явная проблема — одно минимальное решение, применение, тестирование, повторение.

> Патч ДОЛЖЕН придерживаться правил стиля кода проекта (style guidelines), если они определены.

Это просто здравомыслие. Я потратил время на очистку чужих патчей, потому что они настаивали на том, чтобы ставить else рядом с if, а не ниже, как того требует Вселенная. Последовательный код выглядит здоровым.

> Патч ДОЛЖЕН придерживаться руководящих принципов «Разработка публичных Интерфейсов», определенных ниже.

Ах, боль, боль. Я не говорю о том времени, когда мне было восемь лет, и я наступил на доску с торчащем из нее 4-дюймовым гвоздем. Это было еще ничего. Я говорю о 2010-2011 годах, когда у нас было несколько параллельных релизов ZeroMQ, каждый из которых имел разные несовместимые API или проводные протоколы. Это были упражнения в плохих правилах, бессмысленно соблюдаемых, которые и сегодня все еще причиняют нам боль. Правило гласило: «Если вы измените API или протокол, вы ДОЛЖНЫ создать новую основную версию». Проткните мою ногу гвоздем, это менее болезненно.

Одним из больших изменений, которые мы сделали с C4, является запрет подобного санкционированного саботажа. Удивительно, но это даже не сложно. Мы просто не разрешаем нарушать существующие публичные контракты, и точка, если только все не согласятся с этим, тогда да. Как сказал Линус Торвальдс 23 декабря 2012 года: «МЫ НЕ НАРУШАЕМ ПОЛЬЗОВАТЕЛЬСКОЕ ПРОСТРАНСТВО!»

> Патч НЕ ДОЛЖЕН включать нетривиальный код из других проектов, если только Участник не является изначально автором этого кода.

Это правило имеет два эффекта. Во-первых, оно заставляет людей делать минимальные решения, потому что они не могут просто импортировать образцы существующего кода. Из

того, что я наблюдал в других случаях, это всегда приводит к плохим результатам, если только импортированный код не разделен очень четко. Во-вторых, оно устраняет споры по поводу лицензий. Вы пишете патч, вы можете опубликовать его как LGPL, и мы можем принять его. Но если вы найдете фрагмент кода в 200 строк в Интернете и попытаетесь вставить его, мы откажем.

> Патч ДОЛЖЕН четко компилироваться и проходить самотестирование проекта, по крайней мере, на основной целевой платформе.

Для кросс-платформенных проектов справедливо условие, чтобы патч работал в среде разработки, используемой Участником.

> Сообщение коммита ДОЛЖНО состоять из одной короткой (менее 50 символов) строки, в которой задается проблема («Проблема: …»), за которой следует пустая строка, а затем предлагаемое решение («Решение: …»)).

Это хороший формат для сообщений коммита, который подходит для эл. почты (первая строка становится темой, а остальная часть — телом письма).

> «Корректный патч» — это патч, который удовлетворяет вышеуказанным требованиям.

Если вдруг это не понятно, возвращаемся к формулировкам и определениям.

4.6. Процесс разработки

В этом разделе мы поэтапно описываем процесс разработки.

> Изменения в проекте ДОЛЖНЫ регулироваться алгоритмом точного выявления проблем и применения минимальных точных решений этих проблем.

Это эссенция тридцатилетнего опыты разработки программного обеспечения. Это крайне простой подход к разработке: делайте минимальные точные решения реальных проблем, ни больше, ни меньше. В ZeroMQ у нас не было места запросам дополнительных функций. Отношения к дополнительным функциям как к багам смущало некоторых новичков. Но это работало, и не только в open-source. Формулировка проблемы, которую мы пытаемся решить, с учетом каждого отдельного изменения, является главным при принятии решения о том, нужно ли внедрять изменение или нет.

> Чтобы запросить изменения, пользователь ДОЛЖЕН зарегистрировать проблему на Платформе.

Это то, как пользователи разговаривают с участниками. Отслеживайте свои проблемы, чтобы другие могли (возможно) попытаться решить их для вас.

> Пользователь или Участник ДОЛЖНЫ описать проблему, с которой они столкнулись.

«Проблема: нам нужна функция X. Решение: сделать это» — вот так не правильно. «Проблема: пользователь не может выполнять простые задачи A или B, кроме как с помощью сложного обхода. Решение: сделать функцию X» является достойным объяснением. Т.к. каждый, с кем я когда-либо работал, должен был усвоить это, то стоит еще раз повторить: сначала определяйте реальную проблему, а только затем ее решение.

> Пользователь или Участник ДОЛЖНЫ стремиться к консенсусу относительно точности их наблюдения и ценности решения проблемы.

И поскольку многие очевидные проблемы иллюзорны, ясно излагая проблему, мы даем другим возможность исправить нашу логику. «Вы используете только A и B, потому что функция C ненадежна. Решение: сделайте функцию C работоспособной».

> Пользователи НЕ ДОЛЖНЫ регистрировать запросы на новые возможности, идеи, предложения или любые решения проблем, которые явно не задокументированы и не доказуемы.

Существует несколько причин не регистрировать идеи, предложения или запросы функций. По нашему опыту, они просто накапливаются в трекере задач, пока их кто-нибудь не удалит. Лучше, когда мы рассматриваем все изменения как решения проблем, ведь тогда мы сможем трезво расставлять приоритеты. Либо проблема реальна, и кто-то хочет ее решить сейчас, либо ее нет в повестке. Поэтому спискам желаний — нет.

> Таким образом, история версий проекта ДОЛЖНА быть списком значимых проблем, документируемых и решаемых.

Мне бы очень хотелось, чтобы трекер GitHub просто перечислил все проблемы, которые мы решили в каждом релизе. Сегодня нам приходится писать это вручную. Если вы размещаете номер проблемы в каждом коммите, и если использовать трекер GitHub, о который мы, к сожалению, еще не сделали для ZeroMQ, эту историю релизов легче делать механически.

> Чтобы работать над проблемой, Участник ДОЛЖЕН сделать форк репозитория проекта, а затем работать с этой копией.

Здесь мы объясняем модель GitHub fork + pull request, чтобы вновь прибывшим приходилось изучать только один процесс (C4), чтобы стать участником.

> Чтобы отправить патч, Участник ДОЛЖЕН создать Pull Request в проект.

GitHub сделал это настолько простым, что нам не нужно для этого изучать команды git. Иногда я рассказываю людям, которые мне особенно не нравится, что командная строка git

потрясающая, и все, что им нужно сделать, — это детально изучить внутреннюю модель git, прежде чем пытаться использовать ее в реальной работе. Когда я вижу их несколько месяцев спустя, они выглядят… измененными.

> Участник НЕ ДОЛЖЕН производить коммиты непосредственно в проект.

Любой, кто размещает патч, является Участником, и все Участники следуют одинаковым правилам. Никаких особых привилегий для оригинальных авторов, потому что в противном случае мы не создаем сообщество, а только увеличиваем наши эго.

> Чтобы обсудить патч, люди МОГУТ комментировать коммиты и Pull Request'ы на Платформе или в другом месте.

Случайно распределенные дискуссии могут сбивать с толку, но GitHub решает это для всех текущих участников, отправляя электронные письма тем, кто должен следить за тем, что происходит. У нас был тот же опыт и то же решение в Wikidot, и оно работает. Нет никаких доказательств того, что обсуждение в разных местах имеет какой-либо негативный эффект.

> Чтобы принять или отклонить патч, Мейнтейнер ДОЛЖЕН использовать интерфейс платформы.

Работа через веб-интерфейс GitHub означает, что Pull Request'ы регистрируются как проблемы с рабочим процессом и обсуждением. Я уверен, что есть и более сложные способы работы. Все усложнить очень просто, а вот за простотой стоят огромные усилия.

> Мейнтейнер НЕ ДОЛЖЕН принимать свой собственный патч.

Было правило, которое мы определили много лет назад, чтобы остановить выгорание людей: не менее двух человек на проект. Проекты одного человека, как правило, заканчиваются слезами или, по крайней мере, горькой тишиной. У нас довольно много данных о выгорании, почему это происходит и как его предотвратить (даже вылечить). Я расскажу об этом позже в этой главе, потому что, если вы работаете с открытым исходным кодом, вам нужно знать о рисках. Правило «не принимать свой собственный патч» преследует две цели. Во-первых, если вы хотите, чтобы ваш проект был сертифицирован C4, вам нужно взаимодействовать хотя бы с одним человеком, который мог бы помочь. Если никто не хочет вам помочь, возможно, вам нужно переосмыслить свой проект. Во-вторых, контроль за каждым патчем делает его намного более удовлетворительным, удерживает нас в правильном направлении и останавливает нас, если мы нарушаем правила из-за спешки или лени.

> Мейнтейнеры НЕ ДОЛЖНЫ делать оценочные суждения относительно корректных патчей.

Мы уже говорили об этом, но стоит повторить: роль Мейнтейнера заключается не в суждении о сути патча, а только о его технических качествах. Суть ценности патча проявляется только со временем: люди используют его, им он либо понравится, либо нет. А

если никто не использует патч, в конце концов он начнет раздражать кого-то, и его удалят, и никто не будет жаловаться.

> Мейнтейнерам СЛЕДУЕТ быстро принимать исправления.

Существует критерий, который я называю период ожидания изменений, который равен периоду от определения проблемы до тестирования ее решения. Чем быстрее — тем лучше. Если Мейнтейнеры не могут реагировать на Pull Request'ы так быстро, как люди от них того ожидают, значит они не выполняют свою работу (или им нужно больше рук).

> Мейнтейнеры МОГУТ принимать некорректные исправления от других Участников с целью: (а) прекращения бесплодных дискуссий, (б) улавливания неправильных патчей в истории, (в) привлечения Участников к улучшению качества их патчей.

Получается, что быстрое принятие несовершенных патчей, что я называю «оптимистичным слиянием», всегда приводит к лучшим результатам, чем требование от участников идеальной работы.

Обычная практика (пессимистичное слияние, ПС) – ждать, пока не будет окончено длительное интеграционное тестирование (CI), потом выполнить ревизию кода, потом протестировать патч в отдельной ветке, и позже отписать автору отзыв. Автор может исправить патч, и тогда цикл тест/ревизия запускается снова. На этой стадии мейнтейнер может сделать (и часто делает) ценное суждение вроде «мне не нравится, как вы это сделали» или «это не соответствует нашему видению проекта».

В худшем случае патчи могут ждать одобрения неделями, месяцами. Или могут вообще не дождаться. Или они будут отклонены, с какими-нибудь отговорками или доводами.

ПС характерно для большинства проектов, и я уверен, что в большинстве случаев не правильно.

Начну с перечисления проблем, которые создает ПС:

- Оно словно передает своим участникам негативный посыл, который вызывает негативные эмоции: «виновен, пока не доказано обратное». Участники, чувствующие, что им не рады, всегда будут искать альтернативы. А терять участников плохо. Но еще хуже наживать тихих, незаметных врагов.
- Оно дает мейнтейнерам власть над новыми участниками, которой многие из них злоупотребляют. И они могут поступать так на подсознательном уровне. И все же это очень распространено. По своей сути мейнтейнеры будут бороться за то, чтобы оставаться важными в своем проекте. И если они смогут не подпускать потенциальных конкурентов, задерживая и блокируя их патчи, они так и сделают.
- Оно открывает дорогу дискриминации. Кто-то может оспорить это: проект принадлежит своим мейнтейнерам, поэтому они вправе выбирать, с кем работать. Отвечу на это так: не агрессивно инклюзивным проектам суждено погибнуть, и так тому и быть.
- Это замедляет цикл обучения. Инновации требуют быстрых циклов эксперимент-

неудача-успех. Кто-то выявляет проблему или неэффективность продукта. Кто-то предлагает решение. Решение проверяется и либо работает, либо нет. Мы узнали что-то новое. Чем быстрее этот цикл проходит, тем быстрее и более верно продвигается проект.

- Оно дает посторонним возможность троллить проект. Это также просто, как и выдвинуть возражение новому патчу. «Мне не нравится этот код». Обсуждение деталей может потребовать в разы больше усилий, чем само написание кода. Намного легче нападать на патч, чем самому его сделать. Такой баланс благоприятствует троллям и карает честных участников.
- Бремя работы ложится на отдельных участников, что иронично и грустно в open source. Мы хотим работать вместе, но при этом нам говорят править нашу работу самим.

А теперь посмотрим, как все работает при Оптимистичном слиянии (ОС). Для начала необходимо понять, что не все патчи или участники одинаковы. В наших open source проектах мы наблюдали следующие четыре группы:

1. Хорошие участники, которые знают правила и пишут прекрасно, идеальные патчи.
2. Хорошие участники, которые делают ошибки и пишут полезные, но все же битые патчи.
3. Посредственные участники, создающие патчи, которые никто не замечает или не придает значения.
4. Участники-тролли, которые игнорируют правила и которые пишут вредоносные патчи.

ПС утверждает, что все патчи вредоносные, пока не доказано обратное (4). А на самом деле большинство патчей полезны и стоят того, чтобы заняться их улучшением (2).

Посмотрим на сценарии ПС и ОС:

1. ПС: скорость слияния патчей зависит от неопределенных, произвольных критериев. И иногда хороший участник останется с плохим впечатлением. ОС: хорошие участники будут чувствовать себя счастливыми и ценимыми и продолжат делать прекрасные патчи пока не закончат с этим проектом.
2. ПС: участник сдается, правит патч, возвращается словно униженным. ОС: второй участник подключается, чтобы помочь первому отладить их патч. У нас тут короткая, счастливая патч-партия. У нового участника теперь есть помощник и друг в проекте.
3. ПС: мы наблюдаем словесную войну и все удивляются, почему сообщество такое враждебное. ОС: посредственный участник повсеместно игнорируется. Если патч требует доработки, то это произойдет быстро. Участник теряет интерес, и происходит откат патча.
4. ПС: словесная перебранка, в которой побеждают тролли лишь за счет упорства в споре. Общество захлестывают дерись-или-беги эмоции. Продавливаются плохие патчи. ОС: существующий участник сразу откатывает патч. Нет никаких споров. Тролли могут попробовать еще раз, но сразу будут забанены. Вредоносные патчи остаются в git-истории навечно.

В любом случае у ОС результат лучше, чем у ПС.

В большинстве случаев (когда патчи требуют дальнейшей доработки) ОС создает условия

для наставничества и менторства. И мы на самом деле наблюдали это в проектах ZeroMQ, и именно поэтому над ними так весело работается.

> Пользователь, создавший задачу, ДОЛЖЕН закрыть задачу после проверки исправления.

Когда один человек открывает задачу, а другой работает над ней, лучше позволит первому человеку закрыть задачу. Это будет двойной проверкой того, что задача была решена правильно.

> Любой участник, который имеет оценочные суждения о патче, ДОЛЖЕН выразить их через свои собственные патчи.

По сути, целью здесь является позволить пользователям пробовать патчи, а не тратить время в спорах, обсуждая «за» и «против». Насколько легко сделать патч, настолько легко его откатить и применить другой. Вы можете предположить, что это приведет к «войне патчей», но такого не случалось. У нас было несколько случаев в работе с`libzmq`, когда патчи одного участника уничтожались другим участником, который чувствовал, что эксперимент не двигается в правильном направлении. Это легче, чем пытаться достигнуть консенсуса.

> Мейнтейнеры ДОЛЖНЫ закрывать задачи пользователей, которые остаются без действий в течение неприемлемо долгого периода времени.

Держите трекер задач в чистоте.

4.7. Ветки и релизы

Когда работает C4, мы получаем два больших упрощения процесса загрузок. Первый: нам не нужно использовать ветки. Второе, мы загружаем все с мастера.

Это процесс, который мы объясняем в этом разделе.

> Проект ДОЛЖЕН иметь одну ветку («мастер»), которая всегда содержит последнюю версию, и ДОЛЖЕН всегда компилироваться.

Понятно, что каждый патч что-то да добавляет, но не лишне об этом напомнить. Если мастер-ветка не развивается (и проходит свои тесты), кому-то нужно проснуться.

> В проекте НЕ ДОЛЖНЫ использоваться «topic branch» по какой-либо причине. В персональных ветках МОГУТ быть использованы «topic branch».

Вскоре я вернусь к веткам. Вкратце (или «tl;dr», как говорят в интернете), ветки делают репозиторий сложным и разреженным, требуют единогласия – все это дорого, и этого

следует избегать.

> Для создания стабильного релиза, Мейнтейнер должен использовать тэг в репозитории. Стабильные релизы всегда ДОЛЖНЫ быть отделены от мастер-ветки.

4.8. Эволюция публичных контрактов

Под «публичными контрактами» я подразумеваю API и протоколы. До конца 2011 года естественное счастливое состояние`libzmq`было омрачено нарушенными обещаниями и нарушенными контрактами. Мы полностью прекратили давать обещания (т.н. «дорожные карты») для`libzmq`, и наша доминирующая теория изменений теперь заключается в том, что они внедряются внимательно и аккуратно со временем. На встрече в Чикаго в 2012 году Гарретт Смит и Чак Ремес назвали это «пьяной спотыкающейся походкой в сторону величия», так я сейчас об этом думаю.

Мы прекратили нарушать публичные контракты, просто запретив эту практику. Раньше это было «хорошо» (как и в случае с нами, и все горько жаловались, а мы их игнорировали) — ломать API или протокол до такой степени, что нам приходилось менять номер версии. Звучит неплохо, пока вы не получите одновременно находящиеся в стадии разработки версии ZeroMQ 2.0, 3.0 и 4.0, не совместимые друг с другом.

> Все публичные соглашения (API или протоколы) ДОЛЖНЫ документироваться.

Вы думаете, что это было придумано для профессиональных инженеров-программистов, но нет, это не так. Это — правило. Если вы хотите сертификации C4 для своего проекта, убедитесь, что ваши публичные договоренности задокументированы. Никаких отговорок вроде «это указано в коде». Код не является договором. (Да, я намерен в какой-то момент создать процесс сертификации C4, как индикатор качества проектов с открытым исходным кодом).

> Все публичные контракты ДОЛЖНЫ иметь пространство для расширения и экспериментов.

Так, на самом деле общественные договоры меняются. Дело не в том, чтобы не менять их, а в том, что менять их следует безопасно. Это значит обучать (особенно протокольных) разработчиков создавать для этих маневров пространство заранее.

> Патч, который изменяет стабильный публичный договор, НЕ ДОЛЖЕН нарушать работоспособность существующих приложений, если не будет преобладающего консенсуса относительно ценности этого решения.

Иногда патч исправляет плохой API, который никем не используется. Нам нужна свобода, но она должна базироваться на консенсусе, а не догматах одного человека. Однако делать

рандомные изменения «просто потому что» не есть хорошо. В ZeroMQ v3.x разве мы выиграли от переименования ZMQ_NOBLOCK в ZMQ_DONTWAIT? Конечно, это ближе к POSIX сокету recv(), но разве это повод разрушать тысячи приложений? Никто никогда не заявлял это как задачу. Искажение цитаты Столлмана: «ваша свобода создавать идеальная мир заканчивается в дюйме от моего приложения».

> Патч, вводящий новые функции, ДОЛЖЕН делать это с использованием новых имен (новую договоренность).

В ZeroMQ мы раз или два сталкивались с новыми функциями, которые использовали старые имена (или хуже – имена, которые еще использовались где-то). В ZeroMQ v.3.0 был недавно представленный сокет «ROUTER», который был полностью другим, нежели существующий сокет «ROUTER» в 2.x. Господи, фейспалм, почему? Причина: очевидно, даже умных людей иногда стоит контролировать, чтобы они не совершали глупых поступков.

> Новые контракты ДОЛЖНЫ маркироваться как «черновик» («draft»), пока они не станут стабильными и не будут использоваться реальными пользователями.
>
> Старые контракты ДОЛЖНЫ систематически отмечаться как «устаревшие» («deprecated»).

Преимущество этих обозначений жизненного цикла состоит в том, чтобы информировать пользователей о том, что сейчас происходит. «Черновик» означает, что «мы вносим это и намерены оставить, если оно будет работать». Это не значит, что «мы внесли это и уберем в любое время, если пожелаем». Можно считать, что код, который пережил более одного цикла патчей, должен остаться. «Устаревший» значит, что «мы это заменили и намерены это убрать».

> Старые контракты ДОЛЖНЫ систематически отмечаться как «устаревшие» («deprecated») и заменяться их новыми аналогами по мере необходимости.
>
> По прошествии достаточного количества времени, устаревшие контракты ДОЛЖНЫ быть удалены.

В теории это дает приложениям время двигаться в сторону новых стабильных интерфейсов без риска. Вы можете сначала сделать апгрейд, удостовериться, что все работает, а потом со временем доработать все, чтобы устранить зависимость от устаревших и предыдущих протоколов и API.

> Имена устаревших контрактов НЕ ДОЛЖНЫ повторно использоваться новыми контрактами.

Ах, да, помню радость от того, что в ZeroMQ v3.x переименовали топовые функции API (zmq_send[3] и zmq_recv[3]) и выбросили старые названия новых методов, которые были

крайне несовместимы (и которые, я подозреваю, мало кто использовал). Вы, должно быть, опять запутались, ударили себя по лбу, но это реально то, что произошло, и я был также виновен, как и все остальные. Ведь, в конце концов, мы же сменили номер версии! Единственная польза этого опыта была в том, что мы вывели это правило.

4.9. Администрирование проекта

> Учредители проекта ДОЛЖНЫ выступать в качестве Администраторов по набору Мейнтейнеров.

Кто-то должен управлять проектом, и имеет смысл, что учредители должны начать с этого.

> Администраторы ДОЛЖНЫ обеспечить свою собственную преемственность, продвигая наиболее эффективных Мейнтейнеров.

В то же время как учредитель проекта вы на самом деле хотели бы сойти с этого пути прежде, чем станете слишком привязанным к нему. Продвижение самых активных и надежных Мейнтейнеров будет полезным для всех.

> Новый Участник, который делает правильные патчи, который четко понимает цели проекта, и процесс разработки ДОЛЖЕН быть приглашен стать Мейнтейнером.

Повышайте участников быстро, когда видно, что они заслуживают этого. Все остальное контрпродуктивно.

> Администраторы ДОЛЖНЫ отстранять Мейнтейнеров, неактивных в течение длительного периода времени, или неоднократно нарушивших изложенный процесс разработки.

Это было предложение Яна Барбера: нам нужен способ убирать неактивных Мейнтейнеров. Первоначально Мейнтейнеры были самоизбранными, но это затрудняет удаление нарушителей спокойствия (которые редки, но не неизвестны).

> Администраторы ДОЛЖНЫ блокировать «плохих участников», которые вызывают стресс и причиняют боль другим людям, участвующим в проекте. Это должно быть сделано после публичного обсуждения, с возможностью для всех сторон говорить. «Плохой участник» — это тот, кто неоднократно игнорирует правила и культуру проекта, выставляет беспочвенные аргументы, производит враждебные или оскорбительные действия, и который не может самостоятельно корректировать свое поведение, когда другие просят его сделать это.

Время от времени ваши проекты будут привлекать людей неправильного характера. С течением времени вы станете быстрее примечать этих людей. C4 помогает двумя способами. Во-первых, устанавливая строгие правила, он отталкивает искателей хаоса и хулиганов, которые не могут терпеть чужие правила. Во-вторых, это дает вам как Администратору возможность заблокировать их. Мне нравится давать таким людям время, чтобы они могли проявить себя и получать их патчи в публичной записи (причина для слияния плохих патчей, которые, конечно же, можно удалить после подходящей паузы).

Chapter 5. Дизайн, разработка, инновации

> «Размер и разнообразие сообщества является ключевым фактором.»

Давайте рассмотрим инновации, которые Википедия определяет как «развитие новых ценностей посредством решений, которые отвечают новым требованиям, не явным потребностям или потребностям старых клиентов или рынков в новых способах добавления стоимости». На самом деле это значит решать проблемы более дешевым способом. Звучит просто, но истории рухнувших технологических гигантов говорят об обратном. Я постараюсь объяснить, почему команды часто понимают это не правильно, и предложу способ, как нужно создавать что-то инновационное.

5.1. История двух мостов

Два старых инженера разговаривали о жизни и хвастались своими выдающимися проектами. Один из инженеров рассказал, как он разработал один из величайших когда-либо созданных мостов.

«Мы соорудили его над речным ущельем», — сказал он своему другу. «Оно было широким и глубоким. Мы потратили два года, изучая грунт и отбирая проекты и материалы. Мы наняли лучших инженеров и спроектировали мост, на что ушло еще пять лет. Мы наняли крупнейшие инженерные компании для создания структур, башен, контрольных постов и дороги, которые соединили бы мост с основными автомагистралями. Десятки человек умерли в ходе строительства. Ниже уровня дороги мы пустили поезда и проложили специальную дорожку для велосипедистов. С этим мостом связаны годы моей жизни».

Второй человек задумался ненадолго, а потом заговорил. «Как-то вечером мой друг и я, накидавшись водкой, решили перебросить веревку через ущелье», — сказал он. «Просто веревку, привязанную к деревьям. Там было две деревни, по одной на каждой стороне. Сначала люди тянули тюки по этой веревке, используя ремни и шкивы. Потом кто-то перебросил вторую веревку и сделал дорожку, по которой можно пройти. Она была опасной, но дети ее обожали. Потом группа людей переделала ее, сделав более крепкой, и женщины со своими изделиями стали ходить по ней каждый день. По одну сторону моста вырос рынок, который постепенно превратился в крупный город, т.к. там было полно места для домов. Веревочный мост заменили на деревянный, чтобы его могли пересекать лошади и повозки. Потом город построил настоящий каменный мост, с металлическими перекладинами. Позже они заменили камни сталью и сегодня на том самом месте — вантовый мост».

Первый инженер слушал молча. «Забавно», — сказал он — «мой мост был разобран спустя десять лет после того, как мы его построили. Оказалось, что он был построен на неправильном месте, и никто не хотел им пользоваться. Какие-то ребята перебросили веревку над ущельем, несколько миль дальше по течению, и там-то все и ходили».

5.2. Как ZeroMQ потерял свою дорожную карту

Когда я представлял ZeroMQ на конференции Mix-IT в Лионе (Франция) в начале 2012 г.,

меня несколько раз спросили про «дорожную карту». Мой ответ был: нет больше дорожной карты. У нас они были, и мы их ликвидировали. Вместо нескольких экспертов, пытающихся определить следующие шаги, мы позволили событиям развиваться естественным путем. Аудитории не очень понравился мой ответ. Слишком не по-французски.

Однако история ZeroMQ наглядно показывает, почему с дорожными картами возникают проблемы. В начале у нас была маленькая команда разработчиков библиотеки, несколько участников и никакой утвержденной дорожной карты. Поэтому мы собрали воедино наши планы и попытались организовать их в виде релизов. «Вот, — мы писали, — что будет в следующем релизе».

По мере публикации релизов мы столкнулись с проблемой: легко что-то обещать и намного сложнее сделать это в соответствии с планом. Во-первых, большая часть работы была добровольной, и не совсем понятно, как заставить добровольцев следовать дорожной карте. Во-вторых, приоритет задач со временем значительно меняется. Поэтому мы делали обещания, которые не могли сдержать, и выходящие релизы не соответствовали дорожной карте.

Другая проблема заключалось в том, что определяя дорожную карту, мы словно столбили территорию, что затрудняло другим возможность принять участие. Люди предпочитают участвовать в реализации того, что они считают своей идей. Составленный список того, что нужно сделать, больше похож на рутинную работу, а не на интересную возможность поучаствовать.

В итоге мы столкнулись с травматическими изменениями в ZeroMQ, от которых дорожные карты нас не уберегли, несмотря на кучу сил и слов, потраченных на то, чтобы «делать все правильно». Результатом этого стали несовместимые изменения в API и протоколах. Было ясно, что нам нужен новый подход, определяющий процесс внесения изменений. Разработчикам программного обеспечения не нравится идея о том, что мощные, эффективные решения могут появляться без того, чтобы умные разработчики их тщательно выверяли и обдумывали. И все же тогда в Лионе никто не поставил бы под вопрос эволюцию. Это странно и иронично, поэтому далее я еще поразмышляю об этом явлении, ведь на его основе сообщество ZeroMQ развивается с начала 2012 года.

В главенствующей теории инноваций говорится о том, что гениальные индивидуумы, обдумав проблемную ситуацию, выдают точное и аккуратное решение. Иногда у них случаются озарения в стиле «эврика!» — и решение готово. Изобретатель и сам процесс изобретения редкие, драгоценные, единоличные. История знает многих таких героев-одиночек. Мы обязаны им нашим современным миром.

Однако приглядевшись внимательней, вы увидите, что тут кое-что не сходится. История не повествует об одиноких изобретателях. Она рассказывает нам о людях, которым повезло, которые украли или присвоили себе авторство над идеями, появившимися в результате труда многих других людей. Она рассказывает нам о гениальных людях, которые сделали удачный ход, а потом проводили десятилетия в безрезультативных и бессмысленных поисках. Такие крупные и известные изобретатели как Томас Эдисон на самом деле были хороши в систематизации разнообразных исследований, выполняемых большими группами ученых. Это похоже на утверждение о том, что Стив Джобс придумал каждый девайс, созданный Apple. Неплохой миф, хорош для маркетинга, но абсолютно

бессмысленный с практической точки зрения для науки.

История последних десятилетий, которая лучше зафиксирована и которой сложнее манипулировать, наглядно это демонстрирует. Интернет точно является одной из самых инновационных и быстро развивающихся технологий, о становлении которой имеется большое количество достоверной информации. У этой технологии нет изобретателя. Вместо этого есть огромная масса людей, которые тщательно и успешно решали длинную серию текущих проблем, записывали свои ответы и делали их доступными для всех. Инновационная природа Интернета обеспечена не маленькой избранной группой Эйнштейнов. Она обеспечена RFC-документами, которые могут быть использованы и улучшены кем угодно, сотнями и тысячами умных, хотя и не уникально умных людей, программным обеспечением с открытым кодом, который любой может использовать и улучшать. Она происходит из обмена, смешивания и масштабирования сообщества. Она происходит из постоянного увеличения числа хороших решений и избавления от плохих.

Поэтому, вот альтернативная теория инноваций:

1. Есть безграничная область проблем/решений.
2. Область меняется с течением времени в зависимости от внешних обстоятельств.
3. Мы можем точно воспринимать только те проблемы, которые нам близки.
4. Мы можем оценить прибыльность/затратность проблемы, используя рынок решений.
5. Есть оптимальное решение для любой решаемой проблемы.
6. Мы можем достичь этого оптимального решения эвристическим и механическим путем.
7. Наш интеллект может ускорить этот процесс, но не заменить его.

Из этого следует:

- Индивидуальная креативность менее важна, чем процесс. Более умные люди могут работать быстрее, но при этом двигаться в неверном направлении. А коллективное видение реальности помогает нам следовать актуальным путем и не обманывать ни себя, ни других.
- Нам не нужны дорожные карты, если у нас хорошо налажен процесс. Со временем, по мере того, как решения будут конкурировать за долю рынка, функциональность будет расти.
- Мы не столько изобретаем решения, сколько находим их. Соболезнования творческим натурам: творчество является лишь обрабатывающим информацию големом, начищающим до блеска свое собственное эго и озабоченным поднятием кармы.
- Интеллект — это социальный эффект, хотя он и ощущается как что-то личное. Человек, отрезанный от других, перестает думать. Мы не можем ни определить проблемы, ни оценить их решения без других людей.
- Размер и разнообразие сообщества является ключевым фактором. Более крупные и разнообразные сообщества охватывают больше релевантных задач, решают их более точно и делают это быстрее маленькой группы экспертов.

Поэтому когда мы доверяемся экспертам-одиночкам, они делают классические ошибки. Они фокусируются на идеях, а не на проблемах. Они фокусируются на неправильных проблемах. Они делают неправильные выводы о ценности решаемых проблем. И они не пользуются тем, над чем работают.

Можем ли мы применить вышеописанную теорию на практике? В конце 2011 г. я начал документировать C4 и похожие контракты и использовать их в ZeroMQ и в проектах с закрытом кодом. Лежащий в основе процесс я называю «Ориентированной на простоту разработкой», или сокращенно ОПР («Simplicity Oriented Design», SOD). Это воспроизводимый способ разработки простых и элегантных продуктов. Он организует людей в гибкие цепочки поставщиков решений, которые могут быстро и дешево сориентироваться в проблемной области. Они делают это, создавая, тестируя и сохраняя минимальные приемлемые решения, называемые «патчами», или отказываясь от них. Жизнеспособные продукты состоят из длинной череды патчей, применяемых один поверх другого.

Во-первых, ОПР существенна потому, что так мы развиваем ZeroMQ. Она также является базой для процесса разработки, который мы используем при создании крупных приложений ZeroMQ. Конечно, вы можете использовать любую софтверную архитектурную методологию с ZeroMQ.

Чтобы лучше понять то, как мы пришли к ОПР, давайте рассмотрим альтернативы.

5.3. Trash-Oriented Design

Наиболее популярным типом разработки в крупных организациях является «Trash-Oriented Design». TOD основывается на убеждении, что для того, чтобы делать деньги нам нужны крутые идеи. Это упорно всплывающая чушь является мощным костылем для тех, кто лишен воображения. Теория гласит так: идеи редки, поэтому весь фокус в том, чтобы схватить их. Словно далекие от музыки люди восторгаются гитаристом, не понимая, что великие таланты настолько дешевы, что они буквально играют на улицах за копейки.

Основным выхлопом TOD является дорогостоящее «мышление»: концепции, инженерная документация и продукция, которая отправляется прямиком в мусорное ведро. Получается это так: приходят Творческие Люди с длинным списком «мы можем сделать X u Y». Я видел бесконечно детализированные списки всех тех удивительных вещей, которые мог бы делать продукт. Мы все были повинны в этом. Как только свершилась творческая работа по генерации идей, то дело лишь за исполнением их.

Тогда менеджеры и их консультанты передают свои блестящие идеи проектировщикам, которые создают тонны безукоризненно сформулированных документов. Те в свою очередь берут десять лучших идей менеджеров и превращают их в сотню проектов, потрясающих основы бытия.

Эти разработки передаются разработчикам, которые чешут затылок и гадают, кому в голову пришла эта чушь. Они начинают спорить, но ведь проектировщики спустились с Олимпа, и, в конце концов, не смертным разработчикам спорить с творческими людьми и дорогостоящими консультантами.

Тогда разработчики бредут обратно в свои берлоги, униженные, кнутом понуждаемые

строить гигантскую и «очень изящную» рухлядь. И работа эта надрывная, потому что проектировщики не принимают в расчет реальные расходы. Даже мелкие капризы могут обернуться неделями работы. По мере того, как проект замедляется, менеджеры вынуждают разработчиков работать сверхурочно по вечерам и выходным.

В итоге что-то похожее на рабочий продукт видит свет. Это что-то скрипучее, ломкое, сложное и уродливое. Проектировщики клеймят разработчиков за их некомпетентность и платят консультантам еще, чтобы они сделали макияж свинье, и понемногу продукт начинает выглядеть лучше.

К этому времени менеджеры уже начали пытаться продать продукт и обнаружили, неожиданно, что он никому не нужен. Без тени сомнений они смело бросают миллионы долларов на рекламную компанию, объясняющую публике, зачем ей крайне необходим этот продукт. Они заключают сделки с другими организациями, чтобы протолкнуть его на ленивый, глупый и неблагодарный рынок.

После двенадцати месяцев напряженной рекламной компании продукт все еще не приносит прибыли. Хуже того, он драматично терпит неудачи и клеймится прессой как полный провал. Компания потихоньку убирает его на склад, увольняет консультантов, покупает конкурирующий продукт маленького стартапа и называет его версией два своего собственного продукта. Сотни миллионов долларов выброшены на ветер.

А в это время еще один менеджер-визионер где-то там в организации наливает себе точно лишний стаканчик текилы и рассказывает сотрудникам отдела маркетинга о своей Гениальной Идее.

TOD мог бы быть карикатурой, если бы не был так распространен. Около девятнадцати из двадцати продуктов, готовых к выпуску на рынок большими компаниями, ждет провал (да, 87% статистики делается на месте). Из двадцати лишь один возможно преуспеет, да и то благодаря агрессивной рекламе и слабости конкурентов.

Основная мораль TOD ясна, но трудноусвояема: идеи дешевы. Без исключений. Не существует гениальных идей. Любой, кто начинает разговор со словами «О! Еще мы можем сделать вот это!» должен быть побит с рвением странствующих евангелистов. Это тоже самое, что сидеть в кафе у подножия горы, пить горячий шоколад и говорить другим: «Эй, у меня есть классная идея, мы ведь можем взобраться на эту гору! И построить там на вершине шале! С двумя саунами! И садом! Эй, а еще мы можем обеспечить его электричеством с помощью солнечных батарей! Чувак, это же круто! В какой цвет мы его покрасим? В зеленый! Нет, в синий! Ок, идите и сделайте это, а я пока побуду тут и займусь таблицами и графиками!».

Для хорошего начала успешного процесса разработки соберите реальные проблемы, с которыми сталкиваются люди. Вторым шагом будет оценка этих проблем с помощью основного вопроса «Во сколько обойдется решение этой проблемы?». После этого можно сделать список проблем, которые стоит решать. Хорошие решения реальных проблем будут успешным продуктом. Их успех будет зависеть от того, насколько хороши и дешевы решения, и насколько важна проблема (и, к сожалению, насколько большие расходы на маркетинг можно себе позволить). Но их успех будет также зависеть от того, сколько усилий требует их применение, другими словами насколько простыми они будут.

Теперь, сразив дракона абсолютной бесполезности, атакуем демона сложности.

5.4. Complexity-Oriented Design

По настоящему хорошие команды разработчиков и маленькие компании могут обычно заниматься созданием приличных продуктов. Но большая часть продуктов все равно получится слишком сложными и менее успешными, чем могли бы быть. Это все потому, что команды специалистов, даже лучшие из них, часто упрямо практикуют Ориентированную на сложность разработку, (Complexity-Oriented Design, COD), как я ее называю. И работает она так:

- Менеджмент правильно идентифицирует некоторые интересные и сложные проблемы, привлекательные с экономической точки зрения. Вот тут-то они как раз и попадают на колею TOD.
- Команда с энтузиазмом начинает создавать прототипы и работать над ядром. Все это работает, как и было задумано, и команда, загоревшись еще больше, углубляется в напряженную разработку и обсуждение архитектуры, создание элегантных схем, прекрасных и стройных.
- Менеджмент возвращается и воодушевляет команду на решение еще более сложных проблем. Нам свойственно приравнивать затраты к стоимости, поэтому чем сложнее и дороже решение проблемы, тем больше за него можно будет выручить — так им кажется.
- Команда состоит из инженеров, которые любят создавать штуки, и они вступают в дело. Они создают и создают и создают, и кончается все это массивной прекрасно спроектированной сложностью.
- Рынок при знакомстве с продуктом, чешет за ухом и спрашивает: «Что, серьезно, и это лучшее решение, которые вы нашли?». Да, люди используют продукцию, если при этом им не придется тратить свои собственные деньги на подъем на гору мануалов.
- Менеджмент получает позитивные отклики от своих крупных клиентов, которые разделяют мнение о том, что чем выше стоимость (обучения и использования), тем выше ценность, и продолжает толкать процесс.
- В это время где-то в мире маленькая команда занимается решением такой же проблемы, используя лучший подход, и через год разносит сложившееся положение на рынке на мелкие осколки.

Для COD характерны команды, которые одержимы решением неправильных проблем и которые подвержены коллективной мании.

Продукты COD обычно крупные, амбициозные, сложные и непопулярные. Многое из программного обеспечения open source является следствием COD. Для разработчиков безумно сложно остановиться и прекратить расширять проект с целью охватить еще и еще потенциальных проблем. Они спорят: «А что, если кто-то захочет сделать X?», но они никогда не спрашивают себя: «Сколько на самом деле стоит сделать X?».

Хорошим примером COD на практике оказался Bluetooth, сложный, с излишне-усложненной конструкцией комплект протоколов, которые пользователи ненавидят. Он продолжает

существовать только потому, что в сплошь запатентованной отрасли нет реальных альтернатив. Bluetooth прекрасно защищен, что почти бесполезно для бесконтактного протокола. В то же время ему не достает стандартного API для разработчиков, что значит, его реально накладно использовать в приложениях. На канале групповых дискуссий #zeromq участник Wintre однажды написал, как он был взбешен, обнаружив, что в XMMS 2 была рабочая plugin система, но он не мог проигрывать музыку.

COD является кроличьей норой для разработчиков и инженеров, в которой они продолжают и продолжают искать технические решения. Они добавляют все больше и больше функций, закрывая глаза на экономическую сторону их работы.

Основные уроки COD просты, но горьки на вкус:

- Делать что-то, в чем нет необходимости сейчас — бессмысленно. Не важно, насколько вы талантливы или гениальны — если вы занимаетесь тем, что делаете никому не нужные вещи, вы теряете время.
- Проблемы зачастую неравнозначны. Некоторые просто решить, другие сложно. Иронично, но решение простых проблем чаще приносит пользу людям, чем решение сложных проблем. Поэтому если вы позволите вашим разработчикам работать над случайными вещами, скорее всего они сфокусируются на самым интересных, но не актуальных задачах.
- Инженеры и разработчики любят делать разные штуки и украшать их, а это неизбежно приведет к сложности. Крайне важно иметь «стоп-кран», способ задавать короткие, строгие сроки, которые заставят людей искать менее значительные, простые ответы на наиболее важные задачи.

5.5. Simplicity Oriented Design

Наконец, мы подошли к редкой и ценной Ориентированной на простоту разработке (Simplicity Oriented Design, SOD). Этот процесс начинается с реализации: мы не знаем, что мы должны сделать, пока не начнем делать что-то. Выдвижение идей или крупных проектов не просто бесполезно, а мешает разрабатывать по-настоящему точные решения. Действительно лакомые задачи спрятаны, как заветные оазисы, и любая деятельность, кроме разыскивания их, лишь больше окутывает их туманом. Вам нужно быть мобильным, двигаться быстро и налегке.

SOD работает следующим образом:

- Мы составляем список интересных проблем (наблюдая за тем, как люди используют технологию или другие продукты) и располагаем их от простых к сложным, рассматривая и определяя способы использования.
- Мы берем самую простую, самую драматичную проблему и ищем для нее минимальное количество приемлемых решений, или «патчей». Каждый патч решает именно исходную и всеми одобренную проблему наиболее оптимальным способом.
- При оценке патчей мы руководствуемся следующим вопросом: «Можем ли мы найти более простое решение проблемы?» Мы можем измерить сложность количеством концепций и моделей, с которыми пользователю придется ознакомиться или

перебирать наугад для использования патча. Чем меньше, тем лучше. Идеальный патч решает проблему, не требуя ничего от пользователя.

- Развитие нашего продукта заключается в создании патча, который решает проблему «доказательства концепции» и который потом встраивается в единую линию более зрелых продуктов, состоящих из сотен тысяч патчей один поверх другого.
- Мы не делаем ничего, что не являлось бы патчем. Мы принуждаем к этому формальными правилами, которые требуют, чтобы каждое действие или обязанность были привязаны к основной и одобренной всеми задаче, четко сформулированной и задокументированной.
- Мы выстраиваем наши проекты как цепочку поставщиков решений, где каждый проект может обеспечить задачи своим «поставщикам» и получить в ответ патчи. Цепочка поставщиков является «стоп-краном», потому что когда люди нетерпеливо ждут ответа, нам волей неволей приходится работать в узких временных рамках.
- Индивидуумы могут работать над любым проектом и делать патчи для важных по их мнению проблем. Никто из них не «владеет» проектами, они могут лишь принуждать к следованию формальным правилам. У отдельно взятого проекта может быть много вариаций, каждый может обрастать разными патчами, конкурирующими между собой.
- Проекты экспортируют формальные и задокументированные интерфейсы, поэтому проекты-исходники (клиентские) находятся в неведении о проделываемой работе. При этом они могут соревноваться за внимание проектов-клиентов, создавая бесплатный и конкурентный рынок.
- Мы привязываем нашу цепочку поставок к реальным пользователям и внешним клиентам, и мы ведем весь процесс быстрыми циклами с тем, чтобы проблема, полученная от пользователей со стороны могла быть проанализирована, оценена и решена патчем за несколько часов.
- В каждый момент, начиная с первого патча, наш продукт готов к выпуску. Это важно, потому что большая часть патчей будет неправильными (10-30%), и только давая продукт пользователям, мы можем узнать, какие из патчей проблемные и требуют доработки.

SOD — восходящий алгоритм, надежный способ нахождения оптимальных решений наиболее важных проблем в неизведанной области. Вам не нужно быть гением, чтобы использовать SOD, вам просто нужно быть способным видеть разницу между активностью по нагнетанию тумана и прогрессом в решении реальных проблем.

Люди отмечают, что у таких алгоритмов есть ограничения. Можно зациклиться на решении локальных задач. Но так устроена жизнь: собираем маленькие постепенные улучшения длительное время. Не существует гениальных разработчиков. Мы снижаем риск, связанный с локальностью проблем, охватывая всю область, и вообще это спорный вопрос. От ограничений не уйти, они как законы физики. Теория гласит, что

именно так работают инновации, поэтому лучше принять это и работать с этим, а не руководствоваться верой в магию.

Осознав восходящий характер инноваций, вы поймете, почему некоторые команды, компании или продукты застревают в вымышленной стране уменьшающихся перспектив.

У них просто отсутствует разнообразие и коллективная мудрость для нахождения лучших вершин, к которым стремиться. Когда Nokia закрыли свои open-source проекты, они перекрыли себе кислород.

По-настоящему хороший разработчик с хорошей командой может использовать SOD для создания продуктов мирового уровня, быстро и точно. Для максимальной отдачи от SOD разработчик должен использовать продукт длительное время, начиная с первого дня, и развивать свою способность чуять такие проблемы как несогласованность, необычная активность и другие виды неполадок. Нам свойственно не замечать многие досадливые явления, но хороший разработчик обращает на них внимание и находит способ пропатчить их. Суть процесса разработки состоит в исправлении неполадок продукта.

В open source проектах мы делаем эту работу публично. Нет такого момента «а давайте откроем код». Когда так делают, по-моему, это говорит о том, что люди не понимают смысл open source проектов — вовлечь пользователей в ваше исследование и построить сообщество вокруг основной архитектуры.

5.6. Burnout

The ZeroMQ community has been and still is heavily dependent on pro bono individual efforts. I'd like to think that everyone was compensated in some way for their contributions, and I believe that with ZeroMQ, contributing means gaining expertise in an extraordinarily valuable technology, which leads to improved professional options.

However, not all projects will be so lucky and if you work with or in open source, you should understand the risk of burnout that volunteers face. This applies to all pro bono communities. In this section, I'll explain what causes burnout, how to recognize it, how to prevent it, and (if it happens) how to try to treat it. Disclaimer: I'm not a psychiatrist and this article is based on my own experiences of working in pro bono contexts for the last 20 years, including free software projects, and NGOs such as the [http://www.ffii.org FFII].

In a pro bono context, we're expected to work without direct or obvious economic incentive. That is, we sacrifice family life, professional advancement, free time, and health in order to accomplish some goal we have decided to accomplish. In any project, we need some kind of reward to make it worth continuing each day. In most pro bono projects the rewards are very indirect, superficially not economical at all. Mostly, we do things because people say, "Hey, great!" Karma is a powerful motivator.

However, we are economic beings, and sooner or later, if a project costs us a great deal and does not bring economic rewards of some kind (money, fame, a new job), we start to suffer. At a certain stage, it seems our subconscious simply gets disgusted and says, "Enough is enough!" and refuses to go any further. If we try to force ourselves, we can literally get sick.

This is what I call "burnout", though the term is also used for other kinds of exhaustion. Too much investment on a project with too little economic reward, for too long. We are great at manipulating ourselves and others, and this is often part of the process that leads to burnout. We tell ourselves that it's for a good cause and that the other guy is doing OK, so we should be able to as well.

When I got burned out on open source projects like Xitami, I remember clearly how I felt. I simply

stopped working on it, refused to answer any more emails, and told people to forget about it. You can tell when someone's burned out. They go offline, and everyone starts saying, "He's acting strange… depressed, or tired…"

Diagnosis is simple. Has someone worked a lot on a project that was not paying back in any way? Did she make exceptional sacrifices? Did he lose or abandon his job or studies to do the project? If you're answering "yes", it's burnout.

There are three simple techniques I've developed over the years to reduce the risk of burnout in the teams I work with:

- //No one is irreplaceable.// Working solo on a critical or popular project—the concentration of responsibility on one person who cannot set their own limits—is probably the main factor. It's a management truism: if someone in your organization is irreplaceable, get rid of him or her.
- //We need day jobs to pay the bills.// This can be hard, but seems necessary. Getting money from somewhere else makes it much easier to sustain a sacrificial project.
- //Teach people about burnout.// This should be a basic course in colleges and universities, as pro bono work becomes a more common way for young people to experiment professionally.

When someone is working alone on a critical project, you //know// they are going blow their fuses sooner or later. It's actually fairly predictable: something like 18-36 months depending on the individual and how much economic stress they face in their private lives. I've not seen anyone burn-out after half a year, nor last five years in a unrewarding project.

There is a simple cure for burnout that works in at least some cases: get paid decently for your work. However, this pretty much destroys the freedom of movement (across that infinite problem landscape) that the volunteer enjoys.

5.7. Шаблоны для успеха

Это глава с серией шаблонов поведения для достижения успеха в разработке программного обеспечения. Они стремятся включить всё, что отделяет успех от славной трагической неудачи. Они были написаны за один день как “религиозно-маниакальные догматы” руководителем и “всё остальное безумное” — коллегой. Для меня они являются наукой. Но относитесь к Ленивым перфекционистам и другим инструментам так, как вы относитесь к обычным инструментам — заточите их, используйте и выбросите, если подвернется что-то получше.

5.8. Ленивый перфекционист

Никогда не создавайте ничего, что не является точным минимальным решением проблемы, которую мы можем определить и должны решить.

Ленивый перфекционист тратит свое свободное время наблюдая за другими и выявляя задачи, которые нужно решить. Он ищет понимания, всегда спрашивая “В чем реальная проблема?”, затем движется точно и минимально, создавая или заставляя других создавать пригодное для использования решение для одной конкретной задачи. Он использует или поручает другим использовать эти решения, и повторяет это до тех пор, пока не закончатся

нерешенные проблемы, или время и деньги.

5.9. Доброжелательный тиран

Управление большим войском происходит по тому же принципу, что и несколькими людьми, это просто вопрос разделения их на меньшие группы. — Сунь-Цзы

Доброжелательный тиран делит большие проблемы на мелкие и отдаёт их разным группам, чтобы сосредоточиться. Он разбивает задачи между этими группами, как API или решения “вне протокола”, о чем я расскажу в следующей главе. Доброжелательный тиран строит цепочку, которая начинается с проблем и заканчивается нахождением решения. Он безжалостен в том, как работает эта цепочка, но не говорит людям что и как они должны делать.

5.10. Небо и Земля

Идеальная команда состоит из двух частей-сторон: одна для написания кода, другая для обратной связи.

Небо и Земля работают вместе как единое целое, в непосредственной близости, но формально они общаются через решение проблем. Небо получает информацию о проблемах от других пользователей, а также в ходе собственного использования продукта, и “питает” ею Землю. Земля быстро отвечает тестируя решения. Небо и Земля могут коммуницировать через десятки запросов ежедневно. Небо общается с другими пользователями, а Земля — с другими разработчиками. Небо и Земля могут быть двумя разными людьми или двумя небольшими группами людей.

5.11. Открытая дверь

Точность знаний приходит из разнообразия.

Открытая дверь принимает вклад в дело от почти любого. Она не рассуждает о качестве или направлении, взамен позволяя другим поспорить и проявить более активное участие. Она рассчитывает, что даже тролль принесет разнообразие в мнение группы. Она позволяет формировать группе свое мнение о том, что сделает код стабильным, и применяет эти решения с помощью Доброжелательного тирана.

5.12. Смеющийся клоун

Совершенство исключает участие.

Смеющийся клоун, нередко действует как “удачливый неудачник” и не претендует на компетентность. Вместо этого его выходки и неуклюжие попытки провоцируют других на спасение его от собственной трагедии. Так или иначе, он всегда выявляет правильные пути решения проблемы. Люди настолько заняты, доказывая его неправоту, что не замечают, насколько ценную работу проделывают.

5.13. Заботливый генерал

Ничего не планируйте. Разрабатывайте стратегию и тактику, а не ставьте цели.

Заботливый генерал работает на неизведанной территории, решая проблемы, которые скрыты, пока они еще не появились на горизонте. Таким образом, у него нет никаких планов, но он ищет возможности, а затем использует их быстро и точно. Он разрабатывает тактику и стратегию на местах, затем обучает им своих солдат, чтобы те могли двигаться как независимо друг от друга, так и вместе.

5.14. Социальный инженер

Если вы знаете своего врага и знаете себя, вам не нужно бояться и ста сражений. — Сунь-Цзы

Социальный инженер читает сердца и умы тех, с кем он работает. Он спрашивает каждого “Что заставляет тебя сердиться, волноваться, чувствовать себя в безопасности, быть счастливым, аргументировать свою точку зрения или спорить?” Он изучает капризы и предрасположенности. С этими знаниями он может поощрять тех, кто является полезным, и препятствовать тем, кто таковым не является. Социальный инженер никогда не действует основываясь на своих собственных эмоциях.

5.15. Преданный садовник

Тот победит, чья армия воодушевлена единым духом во всех своих рядах.— Сунь-Цзы

Преданный садовник выращивает процесс из маленького семени, шаг за шагом, с каждым новым человеком, приходящим в проект. Он вносит каждое изменение, имея точную причину и согласие ото всех. Он никогда не “спускает причину сверху”, но позволяет другим прийти к консенсусу, а затем обеспечивает соблюдение этого консенсуса. Таким образом, каждый владеет и управляет процессом и управляется в нём: они прикреплены к нему.

5.16. Бродяга

После пересечения реки, вы должны оказаться вдалеке от нее. — Сунь-Цзы

Бродяга принимает свою собственную смертность и скоротечность. У него нет привязанности к своей прошлой работе. Он считает, что всё что мы делаем, окажется в мусоре, это просто вопрос времени. С точными, минимальными вложениями, он может быстро дистанцироваться от прошлого и сосредоточиться на настоящем и ближайшем будущем. Прежде всего он не имеет эго и никакой гордости, поэтому не может пострадать от действий других.

5.17. Пиратская банда

Код, как и все знания, лучше всего работают как частная неколлективная собственность.

Пиратская банда свободно организуется вокруг проблем. Она принимает полномочия постольку, поскольку начальство устанавливает цели и предоставляет ресурсы. Пиратская банда владеет процессом и разделяет его таким образом, что любая задача может быть повторена любым из Банды или передана другому исполнителю. Пиратская банда движется быстро, если возникают новые проблемы, и быстро отказывается от старых решений, если таковые перестают быть актуальными. Ни одно лицо или группа не может монополизировать какую-либо часть цепочки.

5.18. Флешмоб

Вода формирует свой курс в зависимости от грунта, по которому протекает.— Сунь-Цзы

Флешмобы объединяются вместе в пространстве и времени по мере необходимости, а затем эти объединения очень быстро исчезают. Физическая близость имеет большое значение для связи с высокой пропускной способностью. Но со временем это создает технические гетто, где Земля отделяется от Неба. Флешмоб старается собрать много “частых пассажиров”.

5.19. Канарейка-дозорный

Боль, как правило, не является хорошим знаком.

Канарейка-дозорный измеряет качество организации по его собственному уровню страданий и по наблюдаемому уровню удовлетворения тех, с кем он работает. Он приводит новых участников в организации, чтобы тем могли показать еще сырые “страдания невиновных”. Он может использовать алкоголь, чтобы заставить других рассказать о своих болевых точках. Он спрашивает других и самого себя: “Вы счастливы участвовать в этом процессе, и если нет, то почему?” Когда организация процесса причиняет боль ему или другим, он рассматривает это как проблему, которая должна быть решена. Люди должны наслаждаться своей работой.

5.20. Виселица

Никогда не мешай другим совершать ошибки.

Виселица знает, что мы учимся, совершая ошибки, и он накидывает на шею других веревку, чтобы те учились. Он всего лишь аккуратно затягивает веревку, когда приходит время. Немного натяжения, чтобы напомнить другим об их сомнительном положении. Позволяя другим учиться на ошибках, даёт хороший повод чтобы остаться и плохой повод чтобы уйти. Виселица бесконечно терпелив, потому что нет короткого пути, чтобы научиться чему-либо.

5.21. Историк

Сохранение общих записей может быть утомительным, но это единственный способ избежать сговора.

Историк принуждает к публичному обсуждению, чтобы избежать “сговора” на его поле

деятельности. Пиратская банда подразумевает полные и равные коммуникации, которые не зависят от сиюминутного присутствия. Никто не читает архивы, но просто сама вероятность останавливает большинство от злоупотребления. Историк поощряет правильный инструмент для работы: электронная почта для быстрых обсуждений, IRC для болтовни, вики для знаний, а отслеживание ошибок для записи на всякий случай.

5.22. Провокатор

Когда человек знает, что будет повешен через две недели, это невероятно концентрирует мысли.— Сэмуэль Джонсон

Провокатор создает дедлайны, врагов, а иногда и невыполнимое. Команды работают лучше, когда у них нет времени на фигню. Крайние сроки объединяют людей и сосредотачивают коллективный разум. Внешний враг может сподвигнуть пассивную команду к действию. Провокатор никогда не принимает дедлайн слишком серьёзно. Продукт всегда готов к отправке. Но это немного напоминает пропасть с кольями: одна ошибка, и мы все ищем новую работу.

5.23. Мистик

Когда люди спорят или жалуются — просто отправьте им цитату Сунь-Цзы. — Микко Коппанен

Мистик никогда не спорит напрямую. Он знает, что спорить с эмоциональным человеком — только вызывать еще больше эмоций. Вместо этого он уклоняется от дискуссии. Трудно сердиться на китайского генерала, особенно когда он мертв уже 2400 лет. Мистик играет Виселицу, когда люди настаивают на правоте, совершив ошибку.

Chapter 6. Живые Системы

«Живой Системой» называется такая система, которая развивается в естественной среде, самостоятельно приспосабливаясь к новым условиям. Живые Системы могут существовать довольно долгое время, легко адаптируясь к любым изменениям, являясь, таким образом, чрезвычайно эффективными. В отличие от них, “Спланированные Системы” являются, как правило, неустойчивыми, плохо реагирующими на изменения и, как следствие, недолговечными. В этой статье я расскажу о Живой Системе на примере программного обеспечения и общества, а также расскажу о том, как создать подобную систему.

6.1. Почему “Живые Системы”

Согласно Википедии, «Живые Системы» — это сущности, состоящие из самоорганизующихся элементов, активно взаимодействующих с окружающей средой. Эти системы поддерживаются благодаря потокам информации, энергии и веществ.” Данный термин был предложен психологом Джеймсом Гриером Миллером для обозначения концепций жизни.

Я хочу воспользоваться этим термином для создания новой метафоры для систем программного обеспечения и занимающихся ими организаций — двух типов систем, которые представляют для меня наибольший интерес. Эти две системы не просто похожи. Программное обеспечение это продукт, созданный группой людей, и, как отметил Конвэй, **структура системы программного обеспечения отражает структуру организации, которая эту систему разрабатывает.**

Хочу сказать, что “психология программного обеспечения — это психология людей”.

Сегодня большинство программных продуктов хорошо спланированы, но они не становятся «Живыми Системами». Они неумолимо проваливаются на стадии доставки, будучи проданными силой или обманом. Для того, чтобы программное обеспечение стало «Живой Системой», оно должно использоваться организацией, которая его разрабатывает, и тогда оно “живет” или «умирает» вместе с этой организацией. «Организация» может быть чем-то большим, чем компания или команда. Она может включать в себя тысячи команд, предприятий, клиентов и поставщиков, состоящих в необъяснимых, но реально значимых связях.

Ничто не демонстрирует это так наглядно, как интернет, который является «Живой Системой» программного обеспечения, людей, предприятий и других организаций. Организация, создавшая интернет, представляет собой ни что иное как само по себе человеческое общество. Существует множество «Живых Систем», достаточно посмотреть вокруг. Это удивительно простая истина: чем лучше качество создаваемых нами крупномасштабных систем программного обеспечения, тем больше они напоминают окружающую нас реальность.

Существуют также спланированные системы, которые представляют собой полную противоположность Живым Системам. Гораздо легче спланировать систему, чем вырастить ее. Однако планы неминуемо строятся на ложных предположениях и недальновидных решениях. Спланированные Системы в каком-то смысле выглядят привлекательными и

эффективными, однако, они неизбежно терпят поражение. В жизни можно встретить много подобных примеров, скажем, кооперативное хозяйство, спланированные города, Microsoft Windows 8 и так далее.

В программном бизнесе это разделение на живое и спланированное лучше всего отражено в противостоянии свободного и закрытого программного обеспечения. Свободное программное обеспечение (и его брат открытый исходный код) обычно формируется при реальном использовании, в то время как закрытый исходный код обычно спланирован. Это является главной причиной того, почему я не работаю с закрытым исходным кодом: его “смерть” скора и ожидаема. Я предпочитаю, чтобы моя работа сохранялась так долго, как это только возможно.

Я сделаю несколько резких заявлений, начиная с: самые успешные крупномасштабные системы программного обеспечения — “Живые Системы”. Так, в условиях конкурентного рынка, живая система безусловно победит спланированную. Она гораздо быстрее, дешевле и точнее выявит и решит серьезные проблемы. Если в основе вашего бизнеса лежит Спланированная Система, то он уязвим, так как вам не справиться с атаками Живой Системы.

Второе заявление состоит в том, что все вышесказанное также относится к организациям. Если ваша компания представляет собой Спланированную Систему — она уже мертва. В то время как если ваша компания работает как Живая Система, она займет доминирующую позицию на рынке. Интересно то, что, когда две Живые Системы пересекаются, они не конфликтуют. Скорее, они специализируются в разных областях, а затем сливаются, чтобы образовать единую Живую Систему. Конкуренция и конфликт обычно работают на благо Живым Системам, даже если при этом страдают ее отдельные компоненты.

Позвольте мне развить тему конфликта и конкуренции. Конечно, конкуренция, порой даже жесткая, это нормально для людей. Это наша биологическая потребность. Однако в нас также заложена потребность в сотрудничестве, что чаще всего является гораздо более успешной стратегией. Живая Система включает в себя конкуренцию между людьми и сохраняется в случае потери отдельных компонентов. Она, в общем-то, зависит от процесса конкуренции и неудач. Спланированная Система, по сути, пытается действовать индивидуально и не может вынести внутреннюю конкуренцию или потерю отдельных компонентов.

6.2. Что из себя представляют Живые Системы

Живая Система состоит из слабосвязанных компонентов. Она находится вне пространства (таким образом, “распределена”) и времени (таким образом, «асинхронна”). Это значит, что многие процессы происходят в неожиданных местах, в непредсказуемое время. Для главного плановика это представляется опасным хаосом.

В Спланированной Системе, наоборот, время и место ожидаемых событий словно предписаны сценарием. Основное внимание уделяется “управлению и контролю”, где решения принимаются централизованно и сообщаясь со структурой. Спланированные Системы всегда иерархичны, так как такая структура представляет собой наиболее оптимальный способ быстрого распространения информации о принятых решениях от верхов к низам.

Спланированную Систему мы строим, а живая — развивается сама. Моя цель — понять и научить, как искусственно создать живую систему. Я изучаю факторы развития, модели ухода и движущие силы саморазвивающейся системы. На самом деле я говорю о создании искусственной жизни и искусственного разума, но в форме, непривычной для исследователей искусственного интеллекта. Я считаю, что отдельные компоненты — включая вас и меня — не могут быть “умными”, если только в узком и поверхностном смысле: ведь интеллект это свойство, присущее только системам.

Живые Системы характеризуются отсутствием центрального планирования или принятия решений. Посмотрите на проект по разработке программного обеспечения и спросите “кто дизайнер?” Если там есть конкретный дизайнер (а он почти всегда есть) — будь то отдельный человек или организация, то это Спланированная Система. В Живой Системе нет ни дизайнеров, ни планов работы, ни каких-либо определенных планов на будущее, кроме как “выживать и развиваться”.

Живая Система больше похожа на свободный рынок Адама Смита, чем на пятилетку Сталина. Экономика, политика, психология так же важны — наверное, даже более важны — в процессе развития Живой Системы, как и технологии. Свободный рынок зависит от таких ключевых положений как: четкие законы, стандарты, контракты и справедливое управление. Работа Живой Системы зависит от этих же положений.

Управляющий орган принимает законы, которые определяют справедливый рынок, а затем претворяет их в жизнь.Единицы измерения, валюты, контракты и другое. В системах программного обеспечения таковыми законами, например, являются лицензия на исходный код или политика вкладов. Справедливый рынок позволяет любому создавать новое предприятие и конкурировать с другими. Для создания реальной конкуренции (то есть свободного выбора клиентов), клиенты могут запрашивать четкие контракты, которые в программном обеспечении являются задокументированные программные интерфейсы приложений и протоколы.

ДНК Живых Систем это, по сути, набор регулируемых контрактов. Так, интернет развивается из набора Рабочих Предложений (RFCs (protocols called Requests for Comments)), которые регулируются Инженерным Советом Интернета (Internet Engineering Task Force (IETF)). “Живые города” развиваются благодаря уголовному и гражданскому праву, установленных стандартов относительно воды, энергии и отходов, транспорта и так далее.

Если бы все стратегии были честными, необходимости в управлении, регулировании не было бы и вовсе. Однако каждая Живая Система уязвима перед мошенничеством. Есть определенная группа людей, которые обманывают — систематически или в зависимости от ситуации. Зная, как работает рынок, они всегда будут пытаться обернуть ситуацию в свою пользу, даже если это плохо скажется для остальных. Они будут лгать, красть, обманывать, запугивать и принуждать и т.д.

Не проявляя сопротивления подобному мошенничеству, рынок будет страдать и вся система рано или поздно умрет. Централизованная власть — один из способов защиты от мошенничества. Однако и она значительно уязвима: мошенники могут и даже часто захватывают власть сами. В Живых Системах они могут захватить только управляющих, что и происходит постоянно, на самом деле.

При захвате мошенниками управляющих реальной Живой Системы, обычной реакцией является отстранение, по возможности. В открытых исходных кодах систем ПО есть возможность создать ответвление и продолжить работать под лучшим управлением. Вот почему ответвление предоставляет существенную свободу, а не означает поражение. Поскольку ответвление также может использоваться как способ захвата, свободные лицензии (GPL и другие) лучшие для Живых Систем в плане программного обеспечения.

Рост Живых Систем — процесс постоянный и естественный. Это их главная отличительная черта — отсутствие больших затрат усилий на их создание. Здесь же, однако, вы будете наблюдать небольшие изменения. Это может показаться скучным и бесперспективным. Тем не менее, это лучший метод выживания. Живая Система должна делать две вещи. Во-первых, она должна решать ряд вопросов относительно прибыли. Во-вторых, со временем она должна меняться и адаптироваться, следуя за изменениями, происходящими в окружающем в мире.

Что касается Спланированной Системы, то подстроить ее под меняющийся окружающий мир очень сложно, часто даже невозможно. Ресурсы определяют власть. Поэтому Спланированные Системы активно и агрессивно противятся изменениям, отрицают их, а когда без изменений уже не обойтись — они перестают существовать.

Живая же Система получает только выгоду от изменений. Для нее не имеет значения, когда заниматься изучением ландшафта — “сегодня” или “завтра. Она развивается благодаря непрерывному обучению. Чтобы действительно уничтожить ее, вы должны нанести ей большой ущерб, что тяжело сделать, если Живая Система уже успешна и сильно развита.

Для нее работа с небольшими неполадками ничем не отличается от обычной деятельности. На самом деле Живая Система развивается благодаря сложным ситуациям, только если они не являются очень тяжелыми, непреодолимыми. Сложная ситуация это то, что помогает компонентам конкурировать между собой и разрабатывать лучшие решения. То, что не убивает Живую Систему, делает ее только сильнее.

Итак, так как Живые Системы учатся всему и вливаются в новые сферы гораздо быстрее и с выгодой для себя, они будут стремиться к тому, чтобы процветать и доминировать, уничтожая любые конкурентные Спланированные Системы. Они быстро реагируют, перемещая ресурсы в те области, в которых они нужнее. И поскольку им не нужны никакие указания действий, они могут изменяться до любого размера. Отсутствие координирования означает ничем не ограниченный масштаб.

6.3. Компоненты Живой Системы

Давайте обратимся к отдельным компонентам Живой Системы. Помните, что Живая Система похожа на рынок, где компоненты конкурируют за предоставление определенных услуг. Компоненты живой системы обладают определенными чертами, которые отличают их от компонентов спланированных систем. У каждого компонента Живой Системы есть определенная группа владельцы и инвесторы, и за каждым компонентом закреплена отдельная группа (в то время как в Спланированной Системе у каждого компонентов одни и те же владельцы). Компоненты объединяются в сети поставщиков и клиентов, данные, имена и адреса которых всегда доступны для удобства клиента. Легким способом смошенничать является подмена высококачественного компонента низкопробным.

Поэтому управляющему органу возможно придется обеспечить соблюдение идентификации профиля и защитить идентификационные данные.

Компоненты максимально независимы от своего местонахождения. Этот факт создает более крупный и эффективный свободный рынок. Это значит, что мы стремимся к тому, чтобы наша Живая Система была независимой. Это противопоставляется Спланированной Системе, где местоположение играет очень важную роль, а конкуренция между компонентами либо очень мала, либо вовсе отсутствует.

Также компоненты могут совершенно произвольно появляться и исчезать. Нет никаких гарантий того, что компонент, от которого мы зависим сегодня все еще будет существовать или находится в доступе завтра. Вероятно, это кажется ненадежным, но на деле это разумно и обоснованно. Мы не зависим от определенных компонентов, мы полагаемся на контракты. Если нам действительно что-то нужно, перед нами появится множество альтернатив. Если одна из них исчезнет — на смену ей придет другая. Если вы упустите одно такси, вы обязательно поймаете другое.

Компоненты максимально независимы друг от друга. Это значит, что они существуют и изменяются в своем темпе, в своем направлении. Изменение в одном компоненте практически незаметно для другого, разве что через открытый интерфейс. Эта свобода необходима для свободного рынка, движимого специализацией и торговлей. Так, один компонент может сфокусироваться на скорости, а другой — на безопасности.

Поскольку не существует ни общепринятого решения о том, какие компоненты существуют, ни кто их создает, они будут иметь разнородный характер, и это разнообразие компонентов имеет важное значение для понимания всей системы. Набор разнообразных компонентов, задействованный в свободном рынке, справится с решением проблемы быстрее и успешнее, чем сплошная, монолитная Спланированная Система.

Компоненты абстрагированы, что означает, что они могут сами по себе являться целыми системами. Например, веб-адрес может представлять собой отдельную, небольшую часть программного обеспечения (один веб-сервер) или крупную инфраструктуру (интернет-бизнес). В свою очередь, только от владельцев каждой группы зависит, какую систему они будут создавать — Живую или Спланированную. Живая Система сможет благополучно принять в себя компоненты Спланированной Системы. Обратный процесс, однако, невозможен.

Компоненты избегают предварительного соглашения известного как общее изменяющееся состояние. Каждый компонент обладает определенными знаниями, которыми он может делиться с другими, но все они делают это асинхронно. Так, хотя Живая Система и представляет собой большую целостную базу знаний, между компонентами нет никакой гарантированной согласованности. Кажется, это парадоксально. Но разве, скажем, каждый член собрания согласен с повесткой дня?

На самом деле, собрания с их повестками дня и протоколами представляют собой олицетворение общего изменяющегося состояния, от которого зависит Спланированная Система. Спланированные Системы не могут функционировать без систематического предварительного соглашения. При параллельном проектировании ПО мы используем “блокировки” для достижения подобного результата. Доказано, что система ПО,

использующая блокировки для того, чтобы поделиться состоянием компонентов, не будет развиваться. Вы можете попытаться создать распределенное программное обеспечение наподобие Спланированное Системы: поначалу все работает хорошо, но почти или вовсе не растет. В то время как хоть запуск Живой Системы и занимает немного больше времени, ее последующий рост безграничен.

В конечном счете, компоненты являются “ленивыми” и ситуативно-обусловленными. Они работают только тогда, когда есть задачи, требующие решения, и растут и развиваются только тогда, когда для этого есть все новые, выгодные возможности. Это означает, что компоненты могут быть простыми и минималистичными. Кроме того, они могут решить “проблему ландшафта” гораздо более точно, без лишних препятствий и предрассудков. В Спланированной Системе, наоборот, компоненты создаются заранее, исходя из прогнозирования будущих проблем или, в лучшем случае, опыта прошлых.

Пример: на запланированную конференции организаторы выбирают определенные темы, основываясь на опыте прошлого года. Сейчас, за месяц до конференции, очень важное событие привлекло интерес публики к совершенно другой проблеме. Как быстро среагируют организаторы конференции? Конференция, которой управляют участники, может изменяться в режиме реального времени, в то время как запланированной конференции понадобится чуть ли не год, чтобы как-то ответить на это.

6.4. Протоколы Живой Системы

Между компонентами Живой Системы существуют определенные связи. Каждая связь представляет собой комбинацию из потока информации, знаний или обращений, в обоих направлениях. Лучшим способом для моделирования данных отношений являются дискретные события или “сообщения”, которые несут в себе определенный набор связей, взаимоотношений, который мы и называем “протоколами”. В естественных Живых Системах мы также можем наблюдать сообщения и протоколы. Клетки, например, поддерживают связь между собой посредством химических сообщений. Мы, люди, общаемся с помощью набора протоколов, лежащих в основе нашей речи. Например, иерархии, в которых мужчины занимают доминирующее положение, являются характерной особенностью человеческого общества, свидетельствуя о том, что протоколы управления и контроля, на которых эти иерархии основаны, встроены в наши умы, а не познаны. Я могу даже предположить, что мужской разум, руководствующийся нуждой предков организовывать охотничьи кампании, отвечает за Спланированные Системы.

Протоколы имеют много общего между собой. Мы видим протоколы широковещательных передач, где один компонент транслирует сигнал многим слушателям. Такой протокол обычно является односторонним. Обычно обратный сигнал, сигнал от слушателей, не поступает.

Мы видим также протоколы “один к одному”, где два компонента обмениваются знаниями, заданиями, запросами и так далее. Такие протоколы более официальные и в идеале полностью асинхронны. Менее официальные протоколы формируются дольше, создавая, таким образом, всеобщую “задержку”. Если я, например, готовлю, пиццу и я должен узнать про каждый ингредиент, естественно, это займет больше времени. “Вы любите грибы?”, “как насчет чеснока?”, “Хорошо, какой сорт сыра вы предпочитаете?”.

Идеальные отношения, связи направлены на снижение “задержки”, поскольку “задержка” во всей системе являются суммой “задержек” все ее производственно-сбытовой цепочки. Так, если я готовлю себе еду, мне нужно потратить минуту на то, чтобы решить какие-то вопросы относительно пиццы, что добавит минуту к общему времени приготовления. В асинхронном диалоге с малой задержкой, я сразу же задам все вопросы, а над ответами буду думать уже позже, когда они будут поступать мне один за другим.

Для создания эффективных асинхронных систем нам нужны очереди и стратегическое планирование очередности. В идеале, мы всегда сталкиваемся с очередями, когда ожидаем сообщения и стараемся как можно скорее перенаправить их получателю, в целях избежания задержек. Нам необходимы стратегии для работы с полными очередями (пространство не бесконечно): можно просто удалить старые сообщения или приостановить отправку сообщений (только это работает для диалогов “один к одному”, а не для “один ко многим”). Нам может понадобится очереди входящих сообщений, одна за поток, и способность ждать сообщения на этой очереди.

Протоколы — неотъемлемая часть Живой Системы. Они выполняют официальные контракты. Если я спрашиваю “Вы любите чеснок?”, в качестве ответа я ожидаю либо да, либо нет. Разговор о погоде в данном случае будет нарушением контракта. Когда мы развиваем наши Живые Системы, мы должны зафиксировать протоколы, чтобы изучить его и заверить. И чем проще и четче он будет, тем лучше. Сложные, неоднозначные протоколы тяжелы как для изучения, так и для реализации и не вписываются в концепцию свободного рынка.

Некоторые Живые Системы полагаются на доверие и индентификационные номера, и не смотрят на то, заверен ли контракт. Это допустимо, но на очень короткий срок, особенно при обмене знаниями, ведь они тоже уязвимы перед мошенниками. В качестве альтернативы можно обеспечить процесс подтверждения каждого контракта с помощью мета-контрактов.Такая практика часто является даже более продуктивной для торговли. Любой таксист хороший, пока он подвозит нас по правильному адресу и не запрашивает слишком высокую цену. Однако мы ходим получать новости из проверенных источников.

Как только у нас есть контакты, которые можно легко проверить, мы сможем справляться с нарушениями. Если одна стратегия терпит поражение, всегда есть другая. Отказывает другая, можно попробовать следующую. Однако после нарушения контакта, вы вряд ли захотите это продолжать таким образом, так как это может нанести более значимый ущерб.

6.5. Пример из практики: библиотека ZeroMQ и сообщество

ZeroMQ сообщество — это Живая Система людей, которая строит Живую Систему программного обеспечения (подборка программного обеспечения под тем же названием). Хотя я изначально разрабатывал ZeroMQ сообщество с большинством свойств Живой Системы, она вышла только в 2012, отказавшись от услуг своих главных планировщиков.

Это сообщество состоит из слабо связанных проектов, имеющих общую цель, которая заключается в обеспечении очередей или сообщений для других систем ПО. Я утверждал и все еще верю в то, что только Живая Система может оптимально применяться с ZeroMQ.

Проекты ZeroMQ связаны в цепочки поставок официальными отношениями на основе API и wire-протоколами. Не только оформление этих API и протоколов, но и обеспечение контроля их эффективности занимает очень много времени. На самом деле, мы обычно не документируем внутренние компоненты, а только внешние API.

В ней не существует ни централизованного планирования, ни координирования. Однако каждый проект развивается органично, поскольку пользователи вносят в них свои разработки и совершенствуют их. Для того, чтобы сделать этот процесс более простым был создан договор ZeroMQ о сотрудничестве, который гарантирует, что организация будет расширяться, включая в себя всех своих компетентных пользователей.

Любой может начать новый проект ZeroMQ или создать ее новое ответвление для конкурирования и экспериментов. Мы поощряем это, поэтому у нас несколько разных типов конкуренции на разных уровнях. Это хорошо работает на практике. Основными лицензиями являются LGPL v3 или MPL v2, гарантирующие, что ответвления всегда защищены (разработки могут совершаться в обоих направлениях).

Управляющей группой в ZeroMQ сообществе является группа, возглавляемая iMatix, фирмой, которая разработала первое ПО. Особо управлять, в принципе, нет необходимости, за исключением того, чтобы прекратить злоупотребление именем “ZeroMQ”. Четкого документального оформления протоколов достаточно, чтобы клиенты могли проверять своих поставщиков.

ZeroMQ очень хорошо масштабируется. Стоимость добавления нового проекта близка к нулю, не считая затрат на поисковые работы. Проекты асинхронны, они используют пункты из GitHub и запросы на включение кода. Координированием является незначительным или вовсе отсутствует. Мы проверяем код по факту, и исправляем плохой код в процессе следующих разработок, а не обсуждая его.

Полная трансформация ZeroMQ в Живую Систему оказалась сложным процессом, поскольку первоначально не было никаких шансов на успех. Основная часть проектов бесплатного ПО все еще зависит от тщательного планирования. Нарушение стандартных процедур казалось очень странным, если не безумным. Потеря главных вкладчиков — которые предоставляли те полномочия, на которых основывалось центральное планирование, — казалась, в перспективе, катастрофой.

Однако ZeroMQ быстрыми темпами расширилась в пространстве и процветало. Мы опровергли теорию о том, что централизованное планирование очень важно для качества. На самом деле, мы выяснили, что без централизованного планирования программное обеспечение улучшилось по качеству и точности. До этого ZeroMQ была крайне нестабильной, экспериментальной и не отвечала потребностям пользователей, она стала достаточно стабильной, надежной и близкой к тому, чего хотят пользователи.

Сегодня ZeroMQ является примером того, как должна правильно работать Живая Система. Она предоставляет большую ценность как хранилище данных, так как предпринимались многочисленные попытки заменить ее, как предыдущими главными планировщиками, так и другими командами. Примечательно, что каждая Спланированная Система, которая претендовала на то, чтобы быть “лучше, чем ZeroMQ” потерпела крах, тогда как каждая Живая Система, которая начинала конкурировать с ZeroMQ, в конце концов становилась ее

частью.

6.6. Трансформация в Живую Систему

Можно ли превратить Спланированную Систему в Живую? Предположим, что у нас есть техническое право (соглашение от достаточного количества участников или законное право — наличие лицензии); каковы тогда практические требования?

Самым сложным будет получить правильный размер компонентов. Это означает, что придется отбрасывать в сторону уже существующие компоненты и создавать новые. Это может обернуться катастрофой, если сделать подобное со всеми компонентами сразу. Поэтому, при большем количестве компонентов, вы должны начать в одной области, выполнить перепроектирование и потом уже развивать сформировавшуюся в результате культуру.

Размер компонентов обычно зависит от людей, так что подходящий это такой “с которым могли бы работать несколько людей”. Масштаб Живой Системы связан с тем, что туда добавляется все больше компонентов, которые могут использовать и замещать друг друга как угодно, без увеличения своих размеров. Компонент слишком маленький, когда он не может сам по себе обеспечить что- или кого-либо, и слишком большой, когда он не может сфокусироваться на чем-то одном.

И, наконец, вам нужны контракты. Мы получили хорошие результаты для систем ПО, просто приняв контракт ZeroMQ C4.1 использовать его вместе с руководством по стилю программирования и ПО лицензией.

По нескольким причинам я настоятельно рекомендую такую общую лицензию как LGPL (моя теория: если вы пользуетесь слабенькой лицензией как, например, Apache или BSD, у вас точно не получится создать Живую Систему).

Ранее запуск подобной Живой Системы осложнялся тем, что самоорганизующиеся ПО экосистемы не находили надлежащего отражения в документах, да и вообще плохо принимались. Нам не хватало эмпирических данных, демонстрирующих, что такие процессы, как C4.1 могут работать, не говоря уже о том, что могут работать так хорошо. Насколько я знаю, тот контракт был первым контрактом в ПО для Живых Систем.

6.7. Экономика Живых Систем

Как же зарабатывать деньги на свободном программном обеспечении? Мне часто задают этот вопрос. Я всегда даю разный ответ, в зависимости от того, с кем я имею дело — с отдельным человеком, небольшой фирмой, крупной фирмой.

Ключом к пониманию Живых Систем является то, что они, в общем-то, и представляют собой экономику. Ни один компонент не находится в системе просто так. Однако выбор между эгоизмом и альтруизмом — ложная дилемма. В основе Живой Системы лежит и то, и другое. Это базовая теория экономики: будучи эгоистами в специализации и торговле, мы создаем общее благополучие. Это суперспособность человека: масштабная специализация и торговля между отдельными людьми, семьями, поколениями, деревнями, городами и

целыми регионами.

Живая Система принадлежит каждому ее участнику, поэтому ее ценность гораздо сложнее измерить, в то время как Спланированная Система, которой владеют несколько людей из “верхов”, представляет собой определенную видимую ценность как для своих владельцев, так и для сторонних наблюдателей. Однако общая ценность Живой Системы всегда будет превосходить любую конкурирующую ей Спланированную Систему. Живая Система может приносить невероятную прибыль, которая делится между всеми ее участниками.

Вот и первый ответ: **Живая Система может уничтожить конкурирующие Спланированные Системы и тем самым присвоить часть скрытых прежде ценностей.** Мы наблюдаем подобные вещи в реальной жизни: когда свободные рыночные экономики превосходят плановые экономики, что приводит к оттоку квалифицированных работников из последней в пользу первой.

Второй ответ заключается в том, что **мы можем построить новые рынки в успешных Живых Системах, что является невозможным в спланированных.** Хорошим примером этого является интернет: он позволяет создавать новые крупномасштабные экономические проекты, что прежде было невозможно в старых сетях. Эти новые рынки могут быть очень прибыльными.

Спланированная Система может выжить только за счет своих компонентов. Это многим похоже на культ и зависит от таких методов поддержания культа как, например, промывание мозгов, когда немногие процветаю за счет остальных. Спланированные Системы по своей природе неэтичны, а также неустойчивы. Справедливому и свободному рынку неотъемлемо присуща мораль, несмотря на то, что большое количество Спланированных Систем как будто бы представляют рынок.

6.8. Заключение

В этом эссе я рассмотрел искусственные Живые Системы, которые копируют реальные Живые Системы и могут быть созданы по их подобию. Живые Системы находятся вне времени и пространства. Они состоят из большого количества независимых компонентов, которые конкурируют и сотрудничают на свободном рынке услуг, труда, ресурсов и знаний. Эти компоненты возникают и развиваются под давлением рынка независимо друг от друга. Они существуют и сходят на нет в зависимости от того как быстро они могут разрешить те проблемы, с которыми сталкиваются их клиенты.

Компоненты Живой Системы взаимодействуют асинхронно, рассылая сообщения по всей системе, по различным схемам.Эти потоки сообщений в форме протоколов являются обязательными. Чем точнее протокол, тем легче клиентам будет выбрать поставщиков, тем эффективней рынок.

У Живой Системы нет главного владельца, который бы осуществлял контроль, однако, там могут выбираться власти для управления (определения и обеспечения исполнения) контрактами. У нее нет ни одной точки отказа. Вместо того, чтобы воспринимать неполадки, неудачи как что-то экстраординарное или то, чего следует избегать, Живая Система учится на них. Неисправный компонент заменяется на исправный.

Живые Системы развиваются путем обучения, соединяясь в цепи поставок, которые связывают компоненты с внешней средой, окружающим миром. Мы можем измерить эффективность Живой Системы, посмотрев на период ожидания с момента поступления в систему проблемы и до ее разрешения. В Спланированных Системах периоды ожидания могут длиться годами, в хорошо адаптирующихся Живых Системах — несколько часов.

Таким образом, будучи рационально организованными, Живые Системы точно оценивают степень сложности проблемы и объем затрат на нее решение. В отличие от Спланированных Систем, их способы решения проблем основаны на реальных данных, а не на предположениях, догадках и устаревших данных, что позволяет им работать точнее, быстрее и дешевле по сравнению со Спланированными Системами.

Чтобы создать крупномасштабную Живую Систему в программном обеспечении, создайте такую же систему из людей. Они будут сотрудничать, развиваться, правильно функционировать и тем самым доминировать на любом рынке. В то время как конкурирующие Спланированные Системы будут терпеть поражение, разрушаясь, функционируя по отдельности, конкурирующие Живые Системы будут стремиться к тому, чтобы специализироваться в различных сферах, а вследствие сливаться в одну большую единую Живую Систему.

Послесловие

Эта книга рассказывает длинную историю, которая началась, когда я прочитал размышления Столлмана о написании его первого свободного программного обеспечения. В 2005 году мы строили онлайновые сообщества сознательно и агрессивно в политических целях. Тогда мы систематизировали теорию и применяли ее снова и снова. В 2007 году я использовал эти знания для создания большого сообщества для платформы Wikidot.com, в которую я вложил капитал, и был генеральным директором. В 2009 году я использовал эти знания качестве основы для сообщества ZeroMQ, и к 2011 году полностью сжег старые неуклюжие паттерны и заменил их модернизированными современными техниками.

Мы все еще учимся, делая наши процессы более простыми, а наши инструменты лучше. C4 не для всех. Требуется мужество принимать неизвестных вкладчиков и доверять им по умолчанию. Требуется опыт, чтобы понять, что за одной из двадцати улыбок скрыт недоброжелатель. Эти уроки медленно учат нас. Даже с полным руководством вам понадобятся годы чтобы его понять. Поэтому практикуйтесь, будьте готовы часто терпеть неудачу и быть счастливыми. :)

Эта книга - учебник «все в одном» для всех, кто хочет создавать сообщества в Интернете. Он охватывает теорию социальной архитектуры и инструменты, необходимые для создания сообщества. В нем подробно описывается сообщество ZeroMQ, в том числе процесс совместной работы (C4). Это мощная книга для любого, кто создает сообщество с открытым исходным кодом или интернет-сообщество в других областях.

A. Об Авторе

Питер Хинченс - программист, писатель и мыслитель, основавший множество онлайновых сообществ. В 2005 году он возглавил Европейскую борьбу против патентов на программное обеспечение, помогая тысячам активистов организовывать онлайн-акции. В 2007 году он основал успешное и обширное сообщество ZeroMQ.

A.1. Другие книги

Другие книги того же автора: ZeroMQ - Messaging for Many Applications (O'Reilly), Culture and Empire: Digital Revolution (Amazon.com), The Psychopath Code (Amazon.com).

B. Копирайт

На оригинальную книгу

- Edition 1.1: 15 May 2016
- ISBN 978-1533112453

- Published by Pieter Hintjens
- Cover font: Kontrapunkt by Bo Linnemann, Kontrapunkt A/S. Text fonts: EB Garamond by Georg Duffner, MonospaceTypewriter by Manfred Klein.
- http://hintjens.com/books

C. О переводе

Данная книга была переведена и опубликована на ресурсе HabraHabr в блоге компании PhilTech.

Полный перевод книги про построение сообществ: «Социальная архитектура».

Оригинал книги на английском языке.

Переводчики:

- Алексей Стаценко - Координатор проекта
- Сергей Даньшин - Предисловие, Главы 1-5
- Кристина Стрельцова - Стратагемы для успеха open source проектов
- Катя Шихова - Глава 6
- Руслан Ибрагимов - Послесловие