

# Так мы инженеры или просто умельцы?

Архитектура предприятия

Крис Бриттон

Май 2007 г.

## Содержание

[Введение](#)

[Что такое программотехника](#)

[Три характеристики инженерного дела](#)

[Обзор](#)

[Вопросы для критического осмысления](#)

[Материалы для дальнейшего изучения](#)

[Глоссарий](#)

## Введение

Нас посетил менеджер по продажам вместе со своим коллегой из компании, занимающейся системной интеграцией. Они многое рассказали нам о своих процессах, их качестве, уровне зрелости, успешности и наработанном в этой сфере опыте, закончив презентацию емкой, как слоган, фразой: «Мы занимаемся программотехникой». Затем они попрощались и ушли, а у нас внезапно разгорелась дискуссия.

— Программотехника — такая чушь, — высказался менеджер по разработке приложений.

— Только потому, что вы не придерживаетесь наших процессов, — резко парировал менеджер по качеству.

— Нам стоит серьезно присмотреться к гибкой методологии разработки. Там говорится, что разработка программного обеспечения — это искусство, — поучительно произнес менеджер по технологиям.

— Я вас умоляю! — менеджер по разработке приложений терпеть не мог поучений.

— Послушайте, но как можно создать систему бизнес-аналитики, не обеспечив прослеживаемость? — продолжил тему администратор данных, присоединяясь к полемике.

Взаимное непонимание росло. И тут, как всегда, в назревающий конфликт вмешался директор по ИТ и призвал всех к порядку. Затем он обратился ко мне.

— Вы отвечаете за архитектуру, — сказал он, глядя мне прямо в глаза, и я почти физически ощущал, как ему хотелось добавить *и поэтому у вас не слишком много реальной работы*, но вместо этого он произнес. — Вам нужно собрать информацию о программотехнике и сделать доклад на следующем совещании.

## Что такое программотехника

Начнем с рабочего определения. *Программотехника* — это разработка программного обеспечения с применением инженерных дисциплин. Вопросы заключаются в следующем. Что это за дисциплины? Применимы ли они к разработке программного обеспечения? Почему это важно?

Например, инженеры-механики создают технические чертежи с помощью систем автоматизированного проектирования (САПР). Инженеры по разработке программного

обеспечения создают блок-схемы, используют *репозитории* и строят *модели*. (Вернее, некоторые из них это делают.) Не дайте сбить себя с толку словом «модель». Модель — это упрощенное представление реальной или воображаемой системы; все проекты являются моделями, но не все модели являются проектами. Итак, является ли создание модели проектируемой системы инженерным делом? В соборе Св. Павла в Лондоне есть модель собора, созданная для архитектора Кристофера Рена. С помощью этой модели он убедил заинтересованные в проекте стороны принять его проект. Это изумительная большая деревянная модель собора. Ее можно увидеть, [нажав эту ссылку](#)<sup>1</sup>. Модель не предназначена для использования в реальной жизни; то же относится и к моделям, создаваемым для современных проектов по разработке программного обеспечения. Хотя моделирование может быть важным компонентом инженерной дисциплины, тем не менее модель не защитит головы, если пойдет дождь. Само по себе моделирование не является инженерным делом.

Инженеры по строительству мостов должны вести чертежи, и, по целому ряду причин, эти чертежи должны в точности соответствовать реально строящейся конструкции. Таким образом, обеспечивается *прослеживаемость (трассируемость)* от чертежей до реализации. Является ли обеспечение прослеживаемости инженерным делом? Строительные чертежи должны быть точными, поскольку они являются договорными инструкциями для строителей. Строители не имеют права отклоняться от проекта. Более того, они обязаны доказать, что действовали в строгом соответствии с ним. Хотя обеспечение прослеживаемости может быть очень важным компонентом дисциплины строительных работ, в реальности оно не поможет автомобилю пересечь водоем по мосту. Само по себе обеспечение прослеживаемости не является инженерным делом.

С другой стороны, разработка программного обеспечения не является таким ясным заданием, как строительство мостов. Основная причина в том, что *в программировании также допускается проектно-конструкторская работа*. Зачастую требования поступают напрямую к программистам (например, предпочтения по внешнему виду и удобству использования). Программисты используют представившиеся возможности для повышения производительности, им также легче обнаруживать и исправлять ошибки высокоуровневого проектирования. В отличие от них строители мостов редко встречаются со своими заказчиками, имеют меньше возможностей для испытания своих конструкций и редко (будем надеяться) обнаруживают ошибки проектировщиков. В хорошо организованных ИТ-отделах поддержка прослеживаемости между кодом, разработкой и производственными системами является незыблемой традицией.

Однако редко встречается прослеживаемость применительно к моделям проектируемых систем и блок-схемам. В связи с динамичным характером разработки программного обеспечения многие люди считают, что обеспечение прослеживаемости является бременем, которое не оправдывается достигаемыми преимуществами. Разработка программного обеспечения во многом похожа на классические инженерные работы, такие как моделирование и обеспечение прослеживаемости. Но создают ли такие нюансы существенные различия между ними? Я думаю, что да, и могу привести три дополнительных аргумента.

Во-первых, программное обеспечение автоматизирует человеческий замысел, который по своей природе является динамичным, непостоянным и подверженным непрерывным изменениям. Поэтому, как правило, возникает постоянное стремление к внесению существенных изменений в ИТ-приложения после их первоначального выпуска, тогда как конструкция моста не претерпевает существенных изменений после того, как по нему проедет первый автомобиль. Обычно в проектах программного обеспечения нет четких указаний по определению и реализации таких изменений. В отсутствие ценности и полезности разработчики не мотивированы поддерживать проектную документацию в актуальном состоянии, отражающем последние изменения. Поэтому проект и реализация быстро расходятся.

Во-вторых, инженерные проекты трехмерных физических систем, таких как мосты, столь же доступны для понимания в деталях, как и в целом. Например, обычно легко понять, как конкретный несущий трос связан со всей конструкцией моста. Мне это познавательное средство представляется в виде увеличительного стекла, и оно очень полезно для новичков в проектировании. В частности, оно позволяет вписать в контекст ту часть, которую они хотят изменить. А теперь посмотрим, что происходит на практике. Блок-схемы программного обеспечения почти не позволяют «увеличивать и уменьшать масштаб». Компоненты программного обеспечения могут связываться друг с другом в слишком многих измерениях, и их связи зачастую не имеют аналога в физическом представлении. Что еще хуже, их

потенциальные связи с другими компонентами могут быть даже неизвестны на момент проектирования. Современные методики создания блок-схем программного обеспечения не позволяют «увеличивать и уменьшать масштаб».

В-третьих (и, возможно, это важнее всего), люди уже привыкли не верить в точность проектных блок-схем.

Предшествующая дискуссия приводит к общей формуле: инженерное дело подобно планированию. Правда, день высадки союзных войск на Атлантическое побережье Европы во время Второй мировой войны был великолепно спланирован, но это не имело ничего общего с инженерным делом. В качестве завершающего штриха можно было бы добавить к концепции планирования понятие *жизненного цикла разработки программного обеспечения*. Станет ли от этого разработка программного обеспечения аналогична инженерному делу? Представим себе работу художника-модельера, планирующего следующий показ мод. Легко вообразить жизненный цикл разработки одежды: определить тенденции, сгенерировать идеи мозговым штурмом, сделать наброски выбранных идей, разработать подробные эскизы, создать макеты и наладить производство окончательного варианта (схема выдумана для примера; я понятия не имею, как это происходит на самом деле). Станет ли от этого художник-модельер инженером? Вряд ли.

Я зашел к менеджеру по разработке приложений и спросил его, что такое, по его мнению, программистика. Он выгасил здоровенную книгу под названием «Справочник разработчика программного обеспечения» (*Software Engineer's Reference Book*) под редакцией Джона А. Мак-Дермиды, выпущенную издательством Butterworth-Heinemann в 1991 г. Во втором абзаце вводной части говорилось: «Чтобы программистика стала настоящей инженерной дисциплиной, она так же должна быть основана на науке и математике...». В третьем абзаце: «Компьютерные и программные системы обычно моделируются с помощью дискретной математики, и научная основа для этой дисциплины включает в себя теорию автоматов и теорию формальных языков...». В наше время разработчики программного обеспечения мало занимаются математикой, даже если они называют себя инженерами по разработке программного обеспечения; многих из них никогда не слышали о теории автоматов. Является ли математика ключевым фактором превращения разработки программного обеспечения в инженерную дисциплину?

В вычислительной обработке данных есть научная традиция математического доказательства. Это доказательство полноты и правильности программ. Но большая проблема в том, что эти доказательства слишком пространственные и плохо доступны для понимания. Кроме того, для доказательства необходимо продемонстрировать, что программа соответствует требованиям, а для этого придется сначала переработать требования и представить их на формальном языке. А шаг этот далеко не тривиальный. Даже в тех случаях, когда задача ясно очерчена — например, определить, что делает функция сортировки, — вовсе не легко описать это на формальном языке. В некоторых случаях вообще непонятно, с чего начать. Например, как бы вы выразили на формальном языке такое требование: «Отобразите на веб-странице перечень продукции так, чтобы и пользователь мог его просмотреть, и вы сами могли бы организовать возможность поиска?»

Инженеры-строители не доказывают, что их проекты математически верны. Вместо этого они проверяют свои проекты на соответствие определенным критериям, из-за которых может возникнуть неисправность. Список критериев со временем претерпевает изменения.

Знаменитый пример — [мост Такома-Нэрроуз<sup>2</sup>](#). Этот подвесной мост неожиданно обрушился в 1940 году из-за непредвиденных воздействий ветров. В день катастрофы ветер был не слишком сильный, но он вызвал механический резонанс в конструкции. До того момента инженеры-строители не считали механический резонанс проблемой, в связи с которой нужно проводить испытания; но теперь они это учитывают. Критерии анализа для испытания подвесных мостов расширились. Математика для инженерного дела — это средство проведения испытаний, но не для всех испытаний требуется математика. Испытание в аэродинамической трубе не является математическим методом доказательства, несмотря на то, что инженеры-строители могут попутно использовать математику для масштабирования результатов испытаний на реальные объекты.

Раз уж мы об этом заговорили, что еще не выполняется в инженерном деле? Инженерное дело не гарантирует, что объект будет построен вовремя и в рамках сметы. Яркий пример — тоннель

под Ла-Маншем между Англией и Францией, построенный с превышением сметы и позже намеченных сроков. Инженеры тоннеля под Ла-Маншем не могли гарантировать, что клиенты будут довольны. Об этом же свидетельствует и множество неудачных моделей автомобилей.

Инженерное дело связано с профессиональной этикой, поддерживаемой коллегами, профессиональными сообществами и угрозой судебных исков. Надлежащая тщательность важна, но и это не является сутью инженерного дела. У врачей есть профессиональные организации и угроза судебных исков, но врачи не являются инженерами.

Инженерное дело — это не просто выполнение моделирования. Инженерное дело характеризуется не только прослеживаемостью. Инженерное дело — это не планирование. Инженерное дело не зависит от математического доказательства. Инженерное дело не гарантирует удовлетворенности заказчиков. Инженерное дело существует не только в качестве защиты от угрозы судебных исков. Современная разработка программного обеспечения весьма слабо соотносится со всем вышеупомянутым. Какое же определение дать тогда инженерному делу? И еще более важно: что такое программотехника?

Без инженерного дела можно что-то спроектировать и построить; но инженерное дело позволяет создавать более крупные, более сложные и более безопасные конструкции, чем это было бы возможно без него. Можно построить дом без инженерного дела, но только безумец стал бы строить небоскреб без инженерного дела. Инженерное дело — это искусство масштабирования.

## **Три характеристики инженерного дела**

На мой взгляд, три из вышеупомянутых различных характеристик инженерного дела являются ярко выраженными.

### **В зрелых инженерных дисциплинах обеспечивается полная декомпозиция проекта сверху вниз (от большего к меньшему)**

Как упоминалось ранее, инженерным проектам физических систем свойственна возможность «увеличения и уменьшения масштаба». Требования, являющиеся ограничениями для какого-либо уровня проекта, должны исходить из результатов верхних уровней проекта и, в конечном итоге, от заинтересованных сторон. Инженерное дело не похоже на автоматический аппарат, где с одного конца подавался бы «комплект» требований, а с другого конца выходил бы конечный продукт. Инженерное дело иерархическое и поэтапное, что позволяет специалистам-практикам уверенно переходить от верхних концептуальных уровней к деталям реализации.

При строительстве моста требования к прочности опор зависели бы, прежде всего, от типа строящегося моста. На первом этапе проектирования принимаются решения по типу моста (арочный, подвесной, понтонный), расположению, грузоподъемности и т. п. Только после того как приняты такие высокоуровневые решения, исследуются требования к опорам — размер, вес, особые требования к основаниям и др.

В противоположность этому, почти все модели программного обеспечения — это низкоуровневое моделирование компонентов реализации, где отсутствует верхний уровень! Вспоминается один проект, в котором я должен был выполнить реализацию крупной системы для госучреждений. В какой-то момент мне предоставили логическую блок-схему базы данных. Я увидел рулон бумаги размером два на три фута, заполненный маленькими рамками и линиями, общий смысл которых практически не поддавался пониманию. Хотя я мог разобраться в структуре конкретной таблицы или конкретной связи между таблицами, однако было трудно постичь, как эта таблица вписывается в контекст большой системы.

### **В зрелых инженерных дисциплинах используются методы анализа (которые могут быть математическими или нематематическими) для проверки каждого уровня проекта**

Инженеры, работающие над высокоуровневым проектом моста, могут вычислить нагрузку для каждой опоры, исходя из высокоуровневых требований к пролету моста, грузоподъемности и т. п. Они могут это сделать, не зная подробного проекта опор, и, возможно, до того как опоры будут спроектированы. Фактически результаты этих вычислений нагрузки могут быть

требованием к подробному низкоуровневому проекту каждой опоры. У проектировщиков программного обеспечения дела в этом вопросе обстоят хуже. Построение нижних уровней реализации происходит без тщательного анализа и всех необходимых испытаний. Другими словами, тестирование программ обычно демонстрирует, что программы соответствуют подробным техническим условиям, но не может продемонстрировать, что сами технические условия были верны.

### **В зрелых инженерных дисциплинах есть как явные, так и подразумеваемые требования**

Подразумеваемые требования — это те требования, которые автоматически включаются инженерами как часть их профессиональных обязанностей. В большинстве своем это такие требования, о которых инженеры осведомлены больше своих заказчиков. Например, исходный вариант моста Такома-Нэрроуз показал, что ветры представляют проблему для подвесных мостов. Однако среднестатистический государственный деятель не обязан об этом знать. Инженеры-строители никогда не позволили бы себе сказать после обрушения нового моста: «Мы знали, что из-за ветра могут быть проблемы, но в ваших требованиях ничего про это не указывалось». Подразумеваемые требования, которые инженеры изучили благодаря опыту предшествующих поколений специалистов, дают им возможность с уверенностью продвигаться в масштабировании своих проектов. Однако разработчикам программного обеспечения все еще не хватает накопленных исторических знаний, на основе которых можно определить подразумеваемые требования, такие как приемлемая производительность, допустимые потери данных во время восстановления после ошибок, указания по обеспечению секретности и др. (Какая производительность является приемлемой? Если вы этого не знаете, то как можно ожидать, что это знает коммерческий директор?)

В общем, можно дать точное определение инженерного дела. «Инженеры» по разработке программного обеспечения могут поспорить с этим утверждением, но они были бы в этом неправы. Неопределенность разработки программного обеспечения может не быть характерной чертой задачи создания программного обеспечения. Это может быть отражением нехватки исторических знаний и инноваций в методиках и средствах, которые могли бы способствовать достижению зрелости в дисциплине разработки программного обеспечения.

## **Обзор**

Многие характеристики, часто считающиеся необходимыми для превращения разработки программного обеспечения в инженерную дисциплину, могут с таким же успехом применяться к проектам, не имеющим отношения к инженерному делу. Например:

- качественное управление проектами;
- жизненный цикл разработки;
- моделирование;
- концепция дальнейшего развития.

Все приведенное выше имеет ценность и во многих ситуациях необходимо для успешности проекта. Но все это лишь в определенной степени помогает дисциплинировать искусство, вместо того чтобы превратить искусство в инженерное дело.

Инженерное дело обладает следующими характерными чертами.

- Проектирование сверху вниз (от большего к меньшему). В противоположность этому, проект программного обеспечения почти всегда является низкоуровневым, и часто бывает трудно увидеть, как работает система в целом.
- Все проекты (высокоуровневые и подробные) анализируются или испытываются для проверки их правильности. В противоположность этому, разработка программного обеспечения почти полностью полагается на тестирование только продукта, т. е. программного кода.
- Подразумеваемые требования. В инженерном деле есть определенные базовые требования — например то, что мост не должен упасть, — и эти требования не обязательно должны указываться заинтересованными сторонами.

Инженерное дело не гарантирует, что разработка будет выполнена вовремя и в рамках сметы

или что она будет коммерчески успешна. Более того, инженерное дело не является жизненно необходимым для разработок малого и среднего размера с четкими и понятными требованиями. Где инженерное дело действительно необходимо, так это в больших и сложных проектах.

## Вопросы для критического осмысления

Есть два важных вопроса, знать ответ на которые необходимо на любом уровне проекта — от верхнего общего до детального низкого. А именно:

- Как определить, что проект правильный?
- Каковы подразумеваемые ожидания заинтересованных сторон в отношении этой системы?

Заметим также, что по мере продвижения от высокоуровневого к детальному проекту появляется больше заинтересованных сторон. Например, ответственный за администрирование ИТ-систем заинтересуется разработкой только когда архитектура приложения уже готова и пора обсуждать техническую архитектуру. Следовательно, на каждом уровне проектирования следует задавать еще один вопрос: кто входит в число заинтересованных сторон?

## Материалы для дальнейшего изучения

Есть множество книг по управлению проектами, о средах разработки, об архитектуре ИТ и о методиках разработки программного обеспечения. Как уже обсуждалось, в действительности они не имеют отношения к инженерному делу, хотя многие из них претендуют на это.

Углубленное рассмотрение всех этих вопросов представлено в следующем веб-документе, доступном бесплатно:

- Britton, Chris. [Making IT Application Design an Engineering Discipline<sup>3</sup>](#) (Крис Бриттон, «Как превратить проектирование ИТ-приложений в инженерную дисциплину»). Веб-сайт IT Modeller, 2006 г. Здесь более подробно рассматриваются вопросы, связанные с уровнями проектирования и правильностью.

## Глоссарий

**Концепции архитектуры** — способ организации и уточнения задач, связанных с разработкой программного обеспечения. Их основное назначение в том, чтобы гарантировать, что ничего не забыто (например, учтены все точки зрения) и работа не дублируется.

**Модели** — упрощенное представление реальной или воображаемой системы. Все документированные проекты являются моделями, но не все модели являются проектами.

**Репозиторий** — база данных для хранения артефактов проектирования программного обеспечения. Репозиторий и средства управления конфигурацией частично перекрывают друг друга. Вообще говоря, средства управления конфигурацией сосредоточиваются на хранении артефактов, которые используются для определения рабочей или тестовой системы (например, программный код и параметры конфигурации), а репозитории используются для хранения артефактов проектирования для существующих или будущих систем (например, логическая структура базы данных и схемы процессов).

**Жизненный цикл разработки программного обеспечения** — этапы разработки программного обеспечения, структурированные так, чтобы можно было отслеживать переход от одного этапа к другому. Пример жизненного цикла разработки программного обеспечения: начало, проектирование архитектуры, рабочий проект, программирование, тестирование системы, рабочая система. Слово «цикл» в заголовке подразумевает итерационную разработку.

## Об авторе

Крис Бриттон (Chris Britton) ранее работал в архитектурном отделе компании Unisys, а сейчас является независимым консультантом по архитектуре. Большую часть времени он проводит за построением средства моделирования под названием Polyphony. Совместно с Питером Бай (Peter Bye) он стал соавтором второго издания книги *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems* («Архитектуры ИТ и ПО промежуточного слоя: стратегии построения больших интегрированных систем»), издательство Addison-Wesley, 2004 г.

Эта статья была опубликована на *Skyscrapr*, интернет-ресурсе корпорации Microsoft. Дополнительные сведения об архитектуре и архитектурной концепции см. на веб-сайте [skyscrapr.net](http://www.skyscrapr.net)<sup>4</sup>.

### Таблица ссылок

<sup>1</sup><http://www.explore-stpauls.net/oct03/textMM/TriforumGreatModelN.htm>

<sup>2</sup>[http://en.wikipedia.org/wiki/Tacoma\\_Narrows\\_Bridge](http://en.wikipedia.org/wiki/Tacoma_Narrows_Bridge)

<sup>3</sup><http://www.itmodeller.co.uk/papers.htm>

<sup>4</sup><http://www.skyscrapr.net/>

### Содержимое сообщества

© 2011 Microsoft. Все права защищены.