

ГЛАВА 1

Выстраиваем концепцию

В главе 1 начнем рассмотрение темы, как это и положено в инженерном деле, с основ. Если бы нашей темой было программирование, то за основами мы, разумеется, отправились бы в страну математики, возможно даже высшей. Но наша тема — разработка бизнес-приложений, а там, где появляется слово «бизнес», математика переходит в категорию пусть и очень важного, но все-таки инструментария.

Мы, конечно же, не будем рассматривать все теоретические основы бизнеса. На нашей позиции нас более всего интересуют: 1) экономическая модель разработки, точнее, затратная часть этой модели; 2) модель работы на рынке труда ключевых участников процесса разработки, то есть специалиста и его работодателя; 3) те аспекты процесса разработки, которые в современном мире считаются новыми и перспективными (включая влияние на рынок труда). В соответствии с перечисленным выше мы и выстроили содержимое главы.

Главная цель — не просто представить ряд разрозненных соображений и рассуждений на заданную тему, а выстроить их в относительно цельную плотную концепцию. Возможно, полнотой эта концепция похвастаться и не сможет, но совершенно точно должна обладать непротиворечивостью.

Краткий курс политэкономии

К счастью, в современном мире вовсе не обязательно в предисловиях и аннотациях делать реверансы в сторону всесильной и единственно верной теории. Но иногда такой реверанс хочется сделать по доброй воле — из искреннего убеждения, что некая теория, будучи примененной в определенной области нашей с вами деятельности, действительно окажется верной, правильной и почти всесильной.

Наша конкретная область — это разработка бизнес-приложений, что, в свою очередь, является бизнесом. Когда мы говорим «бизнес», мы должны помнить про экономическую теорию (и, понятное дело, наоборот). Экономическая же теория гласит, что для максимально эффективного достижения поставленных целей в условиях ограниченности ресурсов выбор модели поведения в первую очередь обуславливается экономическими факторами. В переводе на русский это означает: «Начиная изучать какой-либо процесс, первым делом сформулируй хотя бы самую простую экономическую модель». Мы не претендуем на полноту формулировки модели, но некоторые ее важные детали постараемся показать, причем наглядно.

Два пути через торфяное болото разработки

Почти как в любом деле, в разработке программного обеспечения можно выделить два принципиально разных подхода (не только на платформе «1С:Предприятие», это универсальный принцип).

Путь 1. Быстро и криво

Простой подход — сделать быстро, затратив минимум ресурсов. Пускай будет слегка криво, пускай не будут учтены нюансы эксплуатации, пускай кое-где вылезут ошибки. Зато заработает прямо сразу (как мы всегда надеемся). Ну а потом исправим ошибки, напишем документацию, отработаем замечания заказчика и сделаем еще десяток добрых и полезных дел.

У такого подхода есть одно серьезное преимущество. Если расположение звезд окажется удачным, а разработчик более или менее опытным — что-то действительно заработает «прямо сразу». И какое-то время это «что-то» будет функционировать.

Но есть и недостатки, не менее серьезные. Во-первых, результат разработки по принципу «быстро и криво» далеко не всегда начинает работать «прямо сразу». Практика показывает, что частенько запуск проходит в строгом соответствии со старой шуткой о промышленном шпионаже: «после сборки тщательно обработать напильником».

Во-вторых, в процессе разработки нас ожидают *типичные ловушки* и, как результат, разработка практически со 100%-ной вероятностью имеет хотя бы один из перечисленных ниже **изъянов**.

- **Неустойчивость к входным данным** («Очень странно, у меня же все загружалось!»). При написании кода использовались самые примитивные

примеры, а в реальных данных заказчика оказалась закопана добрая дюжина грабель с прочными дубовыми черенками.

- **Несоответствие техническому заданию и ожиданиям заказчика в целом** (*«Мы думали, что нужно просто создать и записать документы, а чтобы при повторной загрузке изменять ранее созданные вроде бы речи не было!»*). Прямое следствие тактики «Код пишется параллельно чтению технического задания». Задание читается по диагонали, ведь писать код значительно интереснее.
- **Неполнота реализации требуемых функций** (*«Протоколирование мы пока еще не сделали, и в отчете пока только две колонки без детализации. Мы это потом доделаем, но данные загружать можно уже сейчас»*). Заказчику предлагается что-то вроде легкового автомобиля без стекол и кресел — можно же пока и табуретку поставить. Используя инженерную смекалку, всегда можно найти временное решение. Беда (для заказчика) в том, что *врѐменные* решения с течением *времени* получают статус постоянных, раз уж проверку *временем* выдержали.
- **Наличие элементарных ошибок.** Под «элементарными» здесь понимаются такие ошибки, которые возникают в основном потоке событий приложения, причем с валидными входными данными (*«Здесь пока не обращайтесь внимания, просто нажмите “Отмена”, а дальше сначала нажмите “Обновить”, потом снимите и сразу поставьте обратно эту галочку, а потом уже “Сформировать”»*, — представитель заказчика лихорадочно записывает на бумажку). Работа с установкой «Мы просто пишем код», к сожалению, не предполагает затрат труда и времени на альфа-тестирование хотя бы основного потока событий и уж тем более альтернативных потоков.
- **Низкое качество кода с точки зрения дальнейшего сопровождения и развития функциональности** (через месяц после запуска: *«Нет, эту функцию добавить не получится, тут нужно будет почти все полностью переписать»*. — «Почему?» — «Какие сроки были поставлены, вы помните? Поэтому и переписать!» Через шесть месяцев от представителя другой команды специалистов: *«Кто же это вам такую красоту написал? Руки бы оторвать! Тут нужно все переделать с нуля!»*).

Почему так?

«Ибо дело, свершенное в спешке, никогда не бывает свершено наилучшим образом» (Дэвид Эддингс, «Сияющая цитадель»¹).

¹ Эддингс Д. Сияющая цитадель / Пер. с англ. Т. Н. Кухта. — М.: Эксмо-Пресс, 2001.

Памятка руководителю. Распознать, что разработка велась по принципу «быстро и криво», можно по внешним родовым признакам, даже не заглядывая в код. Вот некоторые из них:

- 1) не менее 50 % трудозатрат – трудозатраты на отладку;
- 2) отсутствие внятных пользовательских инструкций;
- 3) большое (больше двух) количество итераций «сдали результат – попробовали эксплуатировать – что-то не работает – вернули на доработку»;
- 4) наличие ошибок, которые обнаруживаются вторым-третьим кликом;
- 5) наличие замечаний типа «Эта ошибка уже была в одной из предыдущих версий, но ее же исправляли?!»

Нельзя сказать, что подход «быстро и криво» является абсолютным злом. Существует несколько классов задач, где такой подход действительно позволяет значительно сэкономить трудовые и временные ресурсы.

- **Быстрое прототипирование.** Проверить гипотезу, понять принцип действия какого-то механизма платформы, попробовать сопряжение с каким-то сторонним API и т. п. При этом нужно четко понимать, что «быстрый прототип» делается на выброс, это даже не временное решение, а просто пробник.
- **Ликвидация аварий.** При тушении пожара хороши любые средства, и когда система собирается падать (или уже падает), вопросов, насколько грамотно, изящно и эффективно написан «подпирающий» код, не возникает в принципе. Только нужно потом не забыть заменить костыли добротной инженерной конструкцией (см. выше про «временные решения»).
- **Одноразовые инструменты «для себя».** Когда требуется написать, например, какой-то крохотный корректировщик данных вида «использовал и выбросил», абсолютно незачем забивать себе голову «альтернативными потоками событий», «снижением издержек на сопровождение» или другими подобными высокими соображениями.

Иными словами, молоток нужен, чтобы быстро забивать гвозди, но далеко не каждая задача представляет собой гвоздь.

Путь 2. Не спеша, эффективно и правильно

Подход посложнее — не приступать к написанию кода раньше времени, не пытаться сэкономить на ресурсах лишний грошик. Разработка даже самого простого механизма, самой элементарной, казалось бы, функции выстраивается в пять стадий.

1. **Анализ.** Изучение задачи, изучение текущего положения, выявление противоречий, уточнение неочевидных мест, допущений и ограничений.
2. **Проектирование.** Разработка «на бумаге». Принятие основных проектных решений (при необходимости с проверкой на быстром прототипе).
3. **Кодирование.** Собственно написание и отладка кода. Честно говоря, это не самое захватывающее занятие — если все продумано и правильно спроектировано, код является очевидным, и очень жаль, что сам себя он писать пока еще не способен.
4. **Проверка.** Прежде чем передавать какую-то разработку (даже на тестирование), разработчик обязан самостоятельно проверить, по крайней мере, основной поток событий и поток наиболее очевидных ошибок.
5. **Документирование.** Написание инструкций по развертыванию, эксплуатации, разрешению известных проблем. Фиксация «на бумаге» наиболее важных и/или сложных проектных решений.

Кажется слегка громоздким, верно? Но все дело в масштабировании. Если мы пишем простенькую обработку, стадии анализа и проектирования можно пройти за десять минут, не написав ни слова, а инструкция будет представлять собой короткую надпись на форме. Важно, чтобы любая разработка проходила все пять стадий, точно в указанном порядке.

У разработки по принципу «**не спеша, эффективно и правильно**», разумеется, тоже имеются недостатки.

- От разработчика требуется не только мастерство в написании кода, но еще и способности к абстрактному мышлению.
- Также разработчику нужно понимать, что такое бизнес-анализ, и хотя бы мало-мальски уметь мысленно поставить себя на позицию пользователя. Никакой «работы по четкому ТЗ» здесь не будет, любое ТЗ придется потрошить в поисках противоречий и недосказанностей.
- Прямо завтра ничего не заработает. Как говорится, прежде чем поехать, какое-то время будем запрягать.
- Эффективная и вдумчивая разработка кажется значительно дороже быстрой и бездумной. Причем так кажется не какому-то воображаемому критику, а собственному руководству и представителям заказчика.

Преимущества же работы по принципу «**не спеша, эффективно и правильно**» в том, что это позволяет исключить практически все недостатки разработки «быстро и криво»:

- система устойчива к любым входным данным;
- результат соответствует ожиданиям заказчика;
- код содержит минимальное количество ошибок (если и встречаются, то некритические);
- разработка вполне пригодна для сопровождения, а дальнейшее развитие не потребует переписывать все заново.

Смертельная схватка цены и качества

Казалось бы, вопрос только в цене, которую мы (и наш заказчик) готовы заплатить за качество. Достаточно выбрать метод реализации проекта: либо «быстро, криво, дешево», либо «медленно, качественно, дорого». В некоторых случаях упростить выбор заказчику может картинка-мотиватор (рис. 1.1).



Рис. 1.1. Памятка заказчику

Но, правда, есть нюансы.

Закон сохранения стоимости

Дело в том, что любая задача разработки по своей сути является чьей-то проблемой. На «создание проблемы» был затрачен определенный объем труда. Для полноценного решения задачи потребуются такой же объем труда и ни одним нормо-часом меньше.

Здесь работает брат-близнец закона сохранения массы имени Ломоносова и Лавуазье (хотя кто только этот закон не формулировал: в европейской истории среди первых был Эмпедокл, а в мировой — наверняка еще на стенах пещер царапали). Наиболее же остроумную трактовку фундаментального закона, относящуюся к соотношению числа компетентных и некомпетентных работников в компаниях, предложил Скотт Адамс в своей замечательной книге «Принцип Дилберта»¹).

Если поступит установка «сэкономить на кодировании», то разработку, скорее всего, придется вести поспешно, жертвуя качеством. Да, это будет дешево и заказчик заплатит только часть реальной стоимости, но и взамен получит лишь частичное решение своей проблемы — как тот посетитель трактира из старой сказки, которому пришлось довольствоваться ароматом блюд вместо пищи, потому что трактирщику смог предложить лишь звон монет вместо денег.

То, что результат неполный, можно определить по самым разным признакам. Например:

- действия, которые могли быть автоматизированы при полноценной разработке, автоматизированы не все. Следствие — пользователи затрачивают лишнее время на ручные операции;
- не реализована проверка входных данных, нет защиты от неправильных действий пользователей. Следствие — множественные ошибки ввода;
- отсутствуют средства протоколирования. Следствие — невозможно разобраться в причинах некорректного поведения системы («Это пользователь ввел что-то не так или система посчитала неправильно? А может, звезды не так стояли?»); для групповых операций невозможно «откатить обратно» или «запустить заново с места сбоя»;
- добавление новой функции и/или модернизация любой существующей требует затрат, сопоставимых с исходной разработкой;
- большое число ошибок в коде приводит к частым сбоям и простоям (а это тоже затраты заказчика, пускай и косвенные) —

и так далее.

¹ Адамс С. Принцип Дилберта. Взгляд из офисной кабинки на начальство, совещания, причуды дирекции и прочие бедствия / Пер. с англ. Е. Г. Генделя. — Минск: Попурри, 2003.