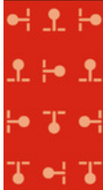
БИБЛИОТЕКА ПРОГРАММИСТА



Сергей Тарасов

ДЕФРАГМЕНТАЦИЯ МОЗГА
СОФТОСТРОЕНИЕ ИЗНУТРИ

- ООП и Ад Паттернов
- NHibernate и SQL
- Архитектуры сокрытия проблем
- Model Driven и процесс

ПИТЕР®

Сергей Тарасов

Дефрагментация мозга. Софтостроение изнутри

Серия «Библиотека программиста (Питер)»

Текст предоставлен правообладателем
http://www.litres.ru/pages/biblio_book/?art=6062542
Дефрагментация мозга. Софтостроение изнутри.: Питер; Санкт-Петербург; 2013
ISBN 978-5-496-00606-4

Аннотация

Эта книга для тех, кто давно связан с разработкой программного обеспечения. Или для тех, кто еще думает выбрать программирование своей профессией. Или для тех, кто просто привык думать и размышлять о происходящем в мире информационных технологий.

Не секрет, что основная масса софтостроения сосредоточена в секторе так называемой корпоративной разработки: от комплексных информационных систем предприятия до отдельных приложений. Поэтому немалая часть сюжетов касается именно Enterprise Programming.

Из текста вы вряд ли узнаете, как правильно склеивать многоэтажные постройки из готовых компонентов в гетерогенной среде, проектировать интерфейсы, синхронизировать процессы или писать эффективные запросы к базам данных. Подобные темы будут лишь фоном для рассказа о софтостроительной «кухне». При определенной доле любопытства вы сможете убедиться, что новое – это хорошо забытое старое, узнать, как устроены некоторые сложные системы, когда следует применять разные технологии, почему специалистам в информатике надо особенно тщательно фильтровать поступающую из множества источников информацию, и многое другое, что вы, возможно, еще не знали или уже знаете, но с другой стороны.

В книге мне хотелось показать наш софтостроительный мир разработки корпоративных информационных систем не с парадного фасада описаний программных сред, подходов и технологий, а изнутри. Насколько это получилось – судить читателю.

Содержание

К читателю 6

О нашей профессии 8

- Очень краткий экскурс 8
- Специализация 11
- Кто такой ведущий инженер, или Как это было 13
- Метаморфозы 17
- О красоте 20
- 6 миллионов на раздел пирога 23
  - Круговорот 25
  - Масштабы и последствия 30
- Профориентация 35
- Начинающим соискателям 38
- Про CV 41
- Про мотивацию 44
- Изгибы судьбы при поиске работы 47

Технологии 51

- Можно ли конструировать программы как аппаратуру? 53
- Безысходное программирование 58
- Эволюция аппаратуры и скорость разработки 60
- Диалог о производительности 65
- О карманных монстрах 67

ASP.NET и браузеры 69
Апплеты, Flash и Silverlight 77
ООП – неизменно стабильный результат 84
Конец ознакомительного фрагмента. 95

Сергей Тарасов
Дефрагментация мозга. Софтостроение изнутри

К читателю

Эта книга для тех, кто давно связан с разработкой программного обеспечения. Или для тех, кто ещё только думает о выборе программирования в качестве своей профессии. Или для тех, кто просто привык думать и размышлять о происходящем в мире информационных технологий.

Не секрет, что основная масса софтостроения сосредоточена в секторе так называемой корпоративной разработки: от комплексных информационных систем предприятия до отдельных приложений. Поэтому немалая часть сюжетов касается именно Enterprise Programming.

В процессе чтения книги вы вряд ли узнаете, как правильно склеивать многоэтажные постройки из готовых компонентов в гетерогенной среде, проектировать интерфейсы, синхронизировать процессы или писать эффективные запросы к базам данных. Подобные темы будут лишь фоном для рассказа о софтостроительной «кухне». При определённой доле любопытства вы убедитесь, что новое – это хорошо

забытое старое, узнаете, как устроены некоторые сложные системы, когда следует применять разные технологии, почему специалистам в информатике надо особенно тщательно фильтровать поступающую информацию и многое другое, что вы, возможно, ещё не знали или знали, но с другой стороны.

В книге мне хотелось показать наш софтостроительный мир разработки корпоративных информационных систем не с парадного фасада описаний программных сред, подходов и технологий, а изнутри. Насколько это получилось – судить читателю.

О нашей профессии

У каждого дела запах особый:
В булочной пахнет тестом и сдобой…

Д. Родари. «Чем пахнут ремёсла?»

Очень краткий экскурс

Более полувека прошло с момента появления первых электронно-вычислительных машин – монстров на базе реле и электронных ламп, занимавших целые здания. Современное умещающееся на ладони устройство во много раз превосходит по вычислительной мощности любого из своих не столь уж дальних предков.

Несколько раз сменилась элементная база, отгремела микропроцессорная революция миниатюризации, изменились технологии, реальностью стали общедоступные вычислительные центры коллективного пользования и глобальная сеть. Неизменной осталась лишь суть профессии программиста. По-прежнему, программист – это человек, способный заставить компьютер решать поставленное перед ним множество задач.

Если первые программисты были сильно ограничены в

средствах и привязаны к аппаратному обеспечению – «железу», к конкретной ЭВМ[1], то современные располагают огромным арсеналом инструментов и технологий, в большинстве случаев позволяющих разработчику не принимать во внимание особенности устройства тех компьютеров, на которых его программа будет выполняться.

С одной стороны, работа, с технологической точки зрения, облегчилась, автоматизировался весь процесс, от написания кода до сборки и компоновки. С другой стороны, требования к желаемому состоянию «задача решена» стали не просто более сложными, но и во многих случаях более неопределёнными. Появилась огромная масса проектов, некритичных к срокам и качеству выполнения. При этом граница применения компьютеров расширилась до областей, казавшихся ранее недоступными. Хотя конструкторы первых ЭВМ скептически относились даже к будущей возможности компьютерной обработки символьной информации.

Джордж Лукас приступил к созданию недостающих серий «Звёздных войн» только через 20 лет. Ещё дольше ждали создатели первых луноходов и автоматических межпланетных зондов, прежде чем выпустить на разведку роботы-марсоходы. Станки с ЧПУ[2], безлюдные заводы и роботы-хирурги не имели практической возможности воплотиться до 1980-

---

[1] Электронная Вычислительная Машина, вдруг кто забыл, как назывались компьютеры в русском языке.

[2] Числовое Программное Управление.

х годов, когда появились массовые достаточно мощные промышленные микропроцессоры. Для многих задач и существующая мощность пока недостаточна, поэтому суперкомпьютеры продолжают её наращивать. Но прогнозы погоды всё равно ошибаются.

Специализация

Исторически сложилось так, что многие программисты были преимущественно математиками и самостоятельно занимались формализацией задач. То есть приведением их к виду, пригодному для решения на компьютере. Сам себе постановщик и кодировщик. В общем виде формула, отражавшая суть работы выражалась так:

Программист = алгоритмизация и кодирование

С ростом сложности задач и упрощения процесса непосредственного кодирования при одновременном усложнении повторного использования кода появилась возможность разделить труд, и формула приобрела примерно такой вид:

Программист минус алгоритмизация = кодировщик
Программист минус кодирование = постановщик задачи

Это вовсе не значит, что мир разделился на аналитиков-алгоритмистов и техников-кодировщиков. Практика многих десятилетий показала, что по-прежнему наиболее востребованным специалистом является инженер, способ-

ный как самостоятельно формализовать задачу, так и воспользоваться стандартными средствами её решения на ЭВМ.

Вот его и следует называть программистом. Современный рынок труда кишит кальками с англоязычных терминов и их комбинациями. Прежде всего, это касается обилия различных видов архитекторов, «девелоперов», «кодёров», «тестеров» и прочих странных прозвищ.

Тестером (не путайте с тостером) раньше назывался прибор-мультиметр, способный измерять напряжение, силу тока и сопротивление. Вы хотели бы работать «тестером»? А если назвать должность в соответствии с её сутью: «инженер-испытатель»? Думаю, отношение к делу сразу изменится.

Чтобы не запутаться в терминологических дебрях жужжащих словечек, сделаем короткое отступление для сопоставления с ранее существовавшей и достаточно понятной всем классификацией.

Кто такой ведущий инженер, или Как это было

Давайте рассмотрим и сравним проектные организации по разработке, поставке и эксплуатации программного обеспечения, оборудования и системной интеграции, условно названные как:

- «А» – организации времён позднего СССР (1960–80 гг.): НИИ[3], КБ[4], КТЦ[5], ВЦ[6]…
- «Б» – современные компании.

Проектные организации в обоих случаях имеют матричную структуру, то есть работники входят в проект безотносительно административного деления, а именно из разных секторов и лабораторий.

Иерархия подразделений

[3] Научно-Исследовательский Институт.

[4] Конструкторское Бюро.

[5] Конструкторско-Технологический Центр.

[6] Вычислительный Центр.

Уровень	А	Б
1	Организация	Компания
2	Отделение (подразделение)	Дивизион (отделение)
3	Отдел (*)	Отдел
4	Лаборатория (**)	Группа
5	Сектор	Группа

(*) – отдел является единицей финансирования, то есть бюджеты составляются, начиная с уровня отдела;

(**) – уровень лаборатории не являлся обязательным для организаций, не ведущих НИР (научно-исследовательскую работу), например для ВЦКП (Вычислительный Центр Коллективного Пользования).

Иерархия должностей

Уровень иерархии	Образовательный ценз (*)	А	Б	Англоязычный термин
1	Доктор (кандидат) наук	Директор	Директор (генеральный директор)	CEO — Chief Executive Officer
1	Доктор (кандидат) наук	Главный инженер	Технический директор	CTO — Chief Technology/Technical Officer
2	Доктор (кандидат) наук	Заместитель по отделению	Заместитель директора по направлению, директор по направлению	(**)
3	Кандидат наук	Начальник отдела	Начальник отдела	Head of service, head of department

4	Кандидат наук	Заведующий лабораторией	Руководитель группы	
5	Высшее образование	Заведующий сектором	Руководитель группы	Team leader
6	Высшее образование	Инженер (категории 3, 2 и 1, старший/ведущий), научный сотрудник (младший, старший)	Специалист, инженер	
7	Среднее специальное образование	Техник, лаборант, студент-практикант	Сотрудник	

(*) – в современных организациях образовательный ценз, как правило, формально не учитывается;

(**) – как правило, укладывается в шаблон Chief «направление» Officer. Например, CAO – Chief Accounting Officer – главный бухгалтер, CDO – Chief Development Of-ficer – директор по развитию и т. д.

Уровни принятия проектных решений

Уровни инвариантны организации, они существуют всегда, но могут быть размытыми, например, если проект небольшой. Используется иерархия деления: система – подсистема – модуль.

Уровень	Тип принимаемых решений
1	Целеполагание, технико-экономическое обоснование, регламент, бюджет, планирование
2	Требования к системе, концепция, техническое задание, архитектура системы
3	Технический проект подсистемы, архитектура подсистемы
4	Реализация подсистемы, спецификация модулей
5	Реализация модулей, ввод информации

Функциональные специализации (роли)

Уровень проектного решения	А	Б	Англоязычный термин
1	Руководитель проекта, научный руководитель	Руководитель проекта	Project Manager
2	ГИП (главный инженер проекта), главный конструктор (*)	Системный архитектор	System Architect
3	Ведущий инженер (область, специализация)	Архитектор подсистемы или направления (**)	Software architect, database architect, hardware architect
4	Инженер (область, специализация)	Разработчик (специализация)	Software engineer, database engineer, system engineer
5	Техник (область, специализация), оператор ЭВМ	Младший специалист, кодировщик	Software developer, database developer, web developer, support & helpdesk

(*) – как правило, главный конструктор или ГИП занимали должности от начальника сектора и выше в зависимости от проекта, который они возглавляли;

(**) – уровень архитектора подсистемы/направления соответствует уровню ведущего инженера. Направления специализации могут быть разнообразными: базы данных, человеко-машинный интерфейс, качество (испытания), информационная безопасность, сетевое оборудование, инфраструктура и т. д.

Метаморфозы

Упомянутое в предыдущей главе разделение на инженеров и техников существовало с незапамятных времён. Застал я его «живьём» в конце 1980-х – начале 1990-х годов. В штате институтского ВЦ или конструкторско-технологического центра всегда присутствовали «инженер-программист» и «техник-программист». Инженеры имели категории вплоть до «ведущего», что при совмещении руководства группой соответствовало нынешнему словечку «тим лид» (*team lead*). Техники тоже делились по категориям.

Формальное различие состояло в том, что техников готовили в профильных профессионально-технических училищах (ПТУ) и техникумах; их образование называлось средним специальным. Инженеров готовили технические вузы[7]. Наконец, были математики-программисты, которых готовили в университетах.

Фактическое же различие состояло в том, что техники не занимались постановкой задач и проектированием программных систем, ограничиваясь непосредственно программированием и эксплуатацией.

Стандартная должность для программиста-техника называлась «оператор ЭВМ». Некоторое время она сохранялась и после перехода на персональные компьютеры как «опе-

---

[7] Высшие учебные заведения.

ратор ПЭВМ». Потом слово трансформировалась в «эникейщик» (от английского *press any key*) и, позднее, «хелпдеск» (*helpdesk*) – специалист службы поддержки пользователей. Соответственно, системный программист-техник большой ЭВМ превратился в системного администратора по эксплуатации сети «персоналок» и серверов.

Современная ситуация изменилась, нередко диплом о высшем образовании, ранее гарантировавший уровень, достаточный для допуска инженера к проектированию, на практике подтверждает лишь уровень техника. Немало средне-специальных учебных заведений за годы вседозволенности, по недоразумению называемой «либерализацией», стало разными университетами и академиями, и наоборот, некоторые инженерные вузы, чей преподавательский состав отдалился от реальных производств и бизнеса, начали выпускать техников с инженерным дипломом. Общее же количество вузов в России с середины 1990-х годов росло как на дрожжах.

Случается наблюдать, как новоиспечённый системой высшего образования программист утверждает: «Мне не понадобилось ничего из того, чем пичкали в институте». Но если эта фраза, произнесенная с гордостью, означает, что работа никчёмная, то эта же фраза, произнесённая с грустью, говорит о том, что вуз – бесполезный. Поэтому следите за интонациями своей речи, делая подобные заявления.

В Европе, и в частности во Франции, для специалистов

с высшим образованием существует чёткое разделение на выпускников инженерных школ и университетов. Первые готовят инженеров для производств, вторые – исследователей для научной и опытно-конструкторской работы. Также негласно считается, что в инженерных школах занимаются серьёзной и целенаправленной подготовкой кадров, тогда как в университетах с их большей внутренней свободой «покуривают травку» между лекциями. Чтобы не объяснять всякий раз новые российские особенности, в результате которых моя альма-матер[8] превратилась из инженерного вуза сначала в академию, а чуть позже и в университет, приходилось в резюме писать прямым текстом «инженерная школа аэрокосмического приборостроения» с устными оговорками о переименовании.

[8] Неформальное название учебных заведений.

О красоте

Споры на тему, является ли программирование или всё софтостроение в целом искусством, ремеслом или чем-нибудь другим, имеют давнюю историю. Как только разработка программ спустилась с академических вершин на грешную землю, появился широкий слой профессионалов, рассматривающих софто-строение, с одной стороны, как средство заработка на жизнь, а с другой – как средство реализации своих идей в техническом творчестве.

Хороший термин – «техническое творчество». От него веет полузабытой атмосферой школьных кружков, где была написана первая программа или смонтировано первое роботоподобное устройство. Те ученики давно подросли, стали профессионалами, но умудрились сохранить творческий подход к делу. А поэтому, как бы ни стандартизировали отрасль, эти люди постараются найти место для реализации своих идей. Сделают не просто «чтобы работало», а чтобы ещё и «было красиво».

Учёный и классик жанра научной фантастики Иван Ефремов писал: «Красота – это высшая степень целесообразности в природе, степень гармонического соответствия и сочетания противоречивых элементов во всяком устройстве, во всякой вещи и во всяком организме».

Нельзя «сделать красиво», если относиться к работе ис-

ключительно утилитарно и шаблонно. Получаются сплошные типовые дома и «мыльные» сериалы. Необходимы и чувство прекрасного, и чувство меры, и знание других образцов, считающихся лучшими. Нужна техническая культура. Долгая работа, неблизкий путь, мотивация преодолеть который исходит, прежде всего, от любви к собственному делу, к профессии.

Но и нельзя «сделать красиво», если рассматривать софтостроение лишь как искусство и средство самовыражения. Любить себя в софтостроении, а не софтостроение в себе. Тогда красота рискует так и остаться не воплощёнными в жизнь эскизами. Невозможно обойтись без знаний технологий производства и хороших ремесленных навыков.

Пока одни корпели над программами и моделями в кружках, другие реализовывали свои интересы, вполне возможно, творческие, но не технические. И вот в связи с процессом заполнения сферы услуг, о котором мы ещё поговорим, эти другие тоже оказались в софтостроении, поскольку имеется устойчивый спрос на рабочую силу. Разумеется, для таких работников главной, а то и единственной целью будет сделать «чтобы как-то работало». Это даже не ремесло в чистом виде, а неизбежные плоды попыток индустриализации в виде халтуры и брака.

В итоге программирование нельзя целиком причислить ни к искусству, ни к ремеслу, ни к науке. Софтостроение на текущий момент – эклектичный сплав технологий, которые

могут быть использованы как профессионалами технического творчества, так и профессионалами массового производства по шаблонам и прецедентам. Поскольку наука все больше отдаляется от софтостроения, то предсказать, что выйдет в каждом конкретном случае – архитектурный шедевр, типовой панельный дом или коровник, практически невозможно. Кадры решат всё.

6 миллионов на раздел пирога

В повести Аркадия и Бориса Стругацких «Гадкие лебеди» есть примечательный фрагмент диалога между писателем Виктором Баневым и школьниками, пригласившими его на встречу:

> – Разрешите мне, – сказал Бол-Кунац. – Давайте рассмотрим схему. Автоматизация развивается в тех же темпах, что и сейчас. Только через несколько десятков лет подавляющее большинство активного населения земли выбрасывается из производственных процессов и из сферы обслуживания за ненадобностью. Будет очень хорошо: все сыты, топтать друг друга не к чему, никто друг другу не мешает. . И никто никому не нужен. Есть, конечно, несколько сотен тысяч человек, обеспечивающих бесперебойную работу старых машин и создание новых, но остальные миллиарды друг другу просто не нужны. Это хорошо?
>
> – Не знаю, – сказал Виктор. – Вообще-то это не совсем хорошо. Это как-то обидно. . Но должен вам сказать, что это все-таки лучше, чем то, что мы видим сейчас. Так что определённый прогресс все-таки налицо.

Со времён написания повести прошло 40 лет, нарисованная школьником схема стала реальностью. Производительность труда растёт, высвобождающиеся из производствен-

ных цепочек люди вынуждены уходить в сферу услуг. Но и она не бездонна. Придумывать и выводить на рынки всё более изощрённые занятия вроде моделирования костюмов для домашних животных становится все труднее.

В Германии совершенно серьёзно и открыто обсуждается идея выплаты всем гражданам безусловного основного дохода[9] (БОД) – минимального пособия примерно в 1000 евро, которого хватит на оплату недорого жилья, скромного питания и одежды. Пособие бессрочное и выплачивается всем гражданам независимо от того, есть у них работа или нет. Те же, кто работает, должны получать заработную плату в качестве добавки к БОД. Во Франции похожее пособие (*Revenu de SolidaritéActive*) существует достаточно давно, с 1988 года, но касается только уже потерявших право на выплаты по безработице; при этом размер пособия порядка 400 евро недостаточен для аренды жилья.

Как определить, любите ли вы свою работу, профессию, дело, которым заняты? Представим на минутку, что вы живёте в Германии и получаете свой БОД. То есть каждому дают по минимальным потребностям, а по способностям – не спрашивают. Ответьте себе честно. Имея возможность потреблять, не отдавая взамен свой труд, останетесь ли вы профессионалом в своей сфере?

Вопрос непраздный. Недалеко от нашего городка есть так называемый «ассоциативный гараж». То есть мастерская об-

---

[9] Оригинальное немецкое название: Bedingungsloses Grundeinkommen.

щего пользования местных авто– и мотолюбителей, где они самостоятельно могут выполнить осмотр и небольшой ремонт своей машины. По субботам в гараж приходят бывшие профессионалы, ныне находящиеся на предпенсионной программе или недавно вышедшие на пенсию. Не секрет, что при европейском качестве жизни к 60 годам многие мужчины всё ещё полны сил и желания работать в своё удовольствие. Они охотно и совершенно бесплатно помогают автолюбителям советами и делом. Для автомастерских в округе подобная ситуация приносит одни убытки. В результате мэрия вынуждена ограничить работу гаража одним днём в субботу до полудня. Если бы такая нелояльная конкуренция продолжалась, то гараж попросту сожгли бы.

Подобная конкуренция среди программистов имеет некоторые отличия, о которых мы и поговорим.

Круговорот

Софтостроение представляет собой соединение относительно небольшого сегмента продуктового производства массово тиражируемых системных сред, средств разработки, прикладных систем, пакетов и огромного рынка услуг, связанных с этими продуктами. Я бы оценил их соотношение как 10 % к 90 %, но, боюсь, такая пропорция будет слишком оптимистичной.

На практике это означает, что на одного производителя

тиражируемого продукта разной степени серийности – от массовых брендов до малотиражных специализированных, приходится почти десяток поставщиков услуг, крутящихся вокруг этих продуктов, и разработчиков заказных программных систем. Чем дальше от основного производителя программных продуктов в мире – США, тем больше это соотношение в пользу сервиса.

Уникальность софтостроения как сферы услуг в его высокой кадровой ёмкости. Представьте себе отдел «X» в управлении крупной фирмы, использовавший для решения какой-то задачи электронные таблицы офисного пакета. На основном производстве тем временем внедрили новую технологию, снизив издержки и сократив несколько работников.

Куда направить освободившиеся ресурсы? А давайте-ка автоматизируем непроизводительную возню отдела «X» с таблицами, и тогда начальники смогут быстрее получать сводки!

Запускается проект. Он не критичен по срокам и качеству. Критичные системы уже давно в эксплуатации, и трогать их никто в здравом уме не будет. Своих программистов в компании нет – непрофильная деятельность. Поэтому заказ спокойно направляют в проектно-консультационную фирму, где, кстати, вполне могут работать выведенные лет 10 назад за штат собственные программисты. У подрядчика, занятого поддержкой существующих систем, может не хватить ресурсов на новый проект, и он вывесит вакансию.

Тем временем уволенные с основного производства приходят в службу занятости, где им говорят: «Специалистов вашего профиля повсюду сокращают. Предлагаем вам переквалификацию». И дружными рядами бывшие операторы устаревшей автоматизированной линии идут на трёхмесячные курсы «Разработка приложений в среде Basic» или «Разработка веб-приложений». После окончания учёбы они попадают на работу в фирму-подрядчик, где начинают автоматизировать использование электронных таблиц отделом компании, откуда их несколько месяцев назад сократили.

Вот такой, если очень упрощенно, происходит круговорот. Возникает естественный вопрос: «А как же конкуренция по себестоимости разработки, которая должна двигать прогресс в отрасли?»

Конкуренция, конечно, формально существует. Но если в производстве она тесно связана со снижением издержек, то в сфере услуг на первый план выходят доверительные отношения между заказчиком и подрядчиком, снижающие риски. Кроме того, проекты нетиповые, заказные, и найти еще одного подрядчика, уже имеющего аналогичный опыт, – это новые затраты и риски. У существующего подрядчика, сопровождающего программы, может быть договор на приоритет новых заказов. Ведь новые программы должны интегрироваться с уже работающими – это опять вопрос отношений с проверенным поставщиком, а попытка сталкивать его лбом с конкурентом может выйти боком вообще всем участ-

никам процесса. Масса скрытых особенностей, непрозрачная среда, полная корпоративных игр и политики. У европейцев есть на сей счёт соответствующая оговорка, что непосредственная власть находится в руках управленцев среднего звена.

На практике замена старого подрядчика новым – весьма рискованная процедура даже на некритичных проектах. Потребуются немалые затраты при неочевидности выгод обмена «шила на мыло». Тогда как менеджеры стремятся, с одной стороны, затраты, наоборот, сократить, а с другой – максимально раздуть бюджет и штат для его освоения ради продвижения по карьерной лестнице. Вот такие противоречивые задачи постоянно вынужден решать менеджер.

Круговорот вовлечения в сферу услуг исключённых из производственных цепочек людей касается не только программистов. За последние два десятилетия практически все крупные западные компании «экстернализировали», то есть вывели за штат, большинство специалистов из отделов информационных технологий. Инфраструктура приложений – серверы, программное обеспечение, администрирование, безопасность – всё поддерживается подрядчиками. В штате остаётся минимум ответственных за связь с подрядчиками и обслуживание парка компьютеров.

Разница в том, что инфраструктурные услуги неплохо оптимизируются и один бывший администратор сети или баз данных компании может теперь обслуживать сразу несколь-

ко клиентов, работая удалённо на прямой связи, а то и непосредственно в ВЦКП[10] или ЦОД[11].

В софтостроении такая оптимизация оказывается проблематичной. Кроме упомянутых проблем с конкуренцией, имеет место и другая веская причина – нечёткость требований, сформулировать которые заказчик далеко не всегда в состоянии. Ведь, как вы помните, он уволил своих прикладных программистов ещё 10–15 лет назад. У подрядчика же функциональная специализация программистов существует только для клиентов, способных давать стабильный заказ с высокой долей прибыли. В первую очередь, это банки и финансовые компании. Проектная фирма обычно вкладывает собственные средства в обучение разработчиков предметной области таких заказчиков, вплоть до получения ими второго высшего образования. Найти же программистов, знающих специфику работы отдела «Х» с электронными таблицами, мягко говоря, маловероятно. Подойдёт и бригада после курсов переквалификации, возглавляемая более опытным руководителем, скорее всего, имеющим не техническое, а коммерческо-управленческое образование.

Мельница крутится, в разработку «проектов для отделов «Х» и следующую за этим через несколько лет переделку втягивается всё больше людей. Можно с уверенностью сказать, что писавших программы в школьных кружках среди

---

[10] Вычислительный Центр Коллективного Пользования.

[11] Центр Обработки Данных (англ. Data Center).

них нет, поскольку такой специалист изначально работал бы в софтостроительной сфере. Хорошо, если они вообще имеют техническое образование. На курсах же дают только некоторый набор приёмов, за счёт которого, постепенно расширяя арсенал, им придётся зарабатывать себе на жизнь. Если голова работает нормально, то бывший новичок за несколько лет превращается в крепкого ремесленника с перспективой сопровождения своих программ до заслуженной пенсии.

Масштабы и последствия

Согласно сведениям IBM, сообщество Java-разработчиков уже к 2006 году насчитывало более 6 миллионов человек[12]. Вдумайтесь в эту цифру. Шесть миллионов ремесленников ежедневно садятся перед монитором и усердно вбивают в дисковое пространство программный код.

Когда вступают в действие большие числа, впору вспомнить о нормальном распределении, на которое нам открыл глаза ещё старина Гаусс.

---

[12] «With today's news, the companies are reinforcing their commitment to the Java community, which comprises more than six million developers worldwide» // IBM Taps Boom in Linux Growth by Expanding Commitment to Partners, Linux and Open Source, december 2005.

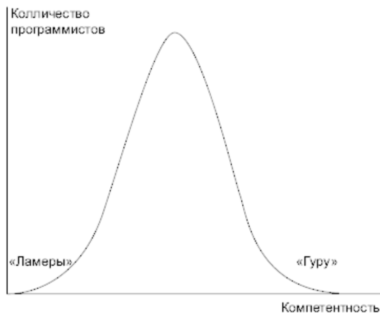


Рис. 1. Нормальное распределение уровня профессиональной компетентности программистов

Чтобы не просто зарабатывать на хлеб, но и мазать его маслом, сохраняя при этом возможности технического творчества, вам лучше держаться подальше от тех направлений деятельности, где конкурентами будут 6 миллионов человек.

И отнюдь не из-за охлофобии. Огромное количество программистов, в первую очередь, означает, что данная технология вполне доступна не только для «середняков», на которых мир держится, но и для откровенных дилетантов. Я даже уверен, что среди дилетантов процент честно заучивающих имена местных «гуру», жаргон и прочие «паттерны» выше, чем среди остальных – для них это, прежде всего, вопрос прохождения интервью.

Большое количество дилетантов нивелирует строчки в резюме и профессиональные сертификаты в глазах заказчика

или работодателя, несмотря на опыт и представленные проекты.

Я не верблюд, чтобы доказывать, что я не верблюд.

Когда выборка составляет 6 миллионов, несложно получить и среднюю по отрасли оплату своего труда. И можно себе представить, скольких усилий стоит добиться высокой оплаты. Не будет иметь большого значения то, что ты можешь сделать хороший дизайн, если за тобой на интервью придёт дилетант, заучивший десять известных работодателю «паттернов», 200 классов фреймворка[13] и просящий за это в 2 раза меньше денег.

Отсюда неутешительный вывод для писавших программы в школьных кружках: количество проектов, где потребуется ваша квалификация, намного меньше количества некритичных заказов, а большинство ваших попыток проявить свои знания и умения столкнется с нелояльной конкуренцией со стороны вчерашних выпускников курсов профессиональной переориентации. На практике это означает, что вам, возможно, придётся снижать цену своего труда и готовиться к менее квалифицированной работе.

---

[13] От англ. framework. В рамках объектно-ориентированного подхода – библиотека классов с двусторонним (взаимным) управлением потоком исполнения программы. Более общее значение – каркас, предоставляющий стандартные службы, библиотеки и компоненты для разработки программ в рамках накладываемых им ограничений.

Не забывайте, что относительная доля критичных к качеству проектов падает, а переделка работающих систем базового уровня, от которых непосредственно зависит бизнес, – и вовсе редкое явление. Этот момент всегда оттягивают до последнего, предпочитая использовать вышедших на пенсию кобол-программистов и модернизацию мейнфреймов[14] с помощью специалистов IBM. Слишком высоки риски. Новые значимые проекты возникают только с новыми рынками и направлениями бизнеса.

Поэтому немало специалистов высокой квалификации уходят в экспертизу и консалтинг, где проводят аудит, обучение, «натаскивание» и эпизодически «вправляют мозги» разным группам разработчиков из числа переквалифицировавшихся.

Да, можно найти проект с уже набившими шишек заказчиками и квалифицированными менеджерами, но места для поиска можно пересчитать по пальцам. И тогда это в принципе мало отличается от работы с узкой специализацией на технологиях. По-прежнему, разработка программ параллельных вычислений, разработка алгоритмов защиты и шифрования или системное администрирование UNIX требуют кадры, которые курсы переквалификации выдать не могут.

Другой доступный вариант – специализация на предметных областях. В этом случае разработчик относительно ав-

---

[14] От англ. mainframe – классическая большая универсальная ЭВМ.

тономен и, во-первых, гораздо менее ограничен в выборе инструментов. Во-вторых, что более существенно, доказывать кому-то степень владения инструментарием у него нет необходимости. К сожалению, хорошее знание предметных областей в сочетании с глубокими техническими знаниями платформ встречается редко в связи с плохой совместимостью высокоуровневых абстракций и низкоуровневых деталей. Обычная эволюция такого специалиста – системный аналитик, сохранивший знания технологий времён своего последнего сеанса кодирования в интегрированной среде.

Хорошо оплачиваемая работа с творческим подходом к труду в современном мире – это привилегия, за которую придётся бороться всю жизнь. И софто-строение здесь не является исключением.

Профориентация

В средней школе многие проходили профориентационные тесты по классификации Климова. Помните, «человек – человек», «человек – техника»? Сколько из вас тогда попало в категорию «человек – человек»? В нашем классе совершенно обычной ленинградской средней школы таковых было менее трети. Немного позднее в классе математической школы тот же самый тест дал ещё меньший результат.

Как вы помните, софтостроение на 90 % находится в сфере услуг. Если вы не работаете на производстве у одного из поставщиков тиражируемого программного обеспечения, то взаимодействия типа «человек – человек» становятся необходимым и важным элементом повседневной работы, если только вы не предполагаете всю жизнь провести в кодировании чужих спецификаций, не всегда толковых и формализованных. Вместо решения сугубо технических задач вроде оптимизации конфигурации версий продукта для разных типов клиентов, вашей целью будет решение задач конкретных клиентов. А критерием решения станет субъективная степень удовлетворённости клиента.

Во французском языке существует специальный термин «чувство службы» (*sens de service*). В русском языке также имеется старинное полузабытое слово «*услужливый*». Чтобы работать софтостроителем в сфере услуг, нужно, прости-

те за тавтологию, уметь быть услужливым. В ещё большей степени «чувство службы» касается консультантов.

Например, когда для публикации функции подключаемого модуля (*plug-in*) в меню основного приложения требуется тем или иным образом её декларировать в семи-восьми разных местах, эксперт-аудитор с «чувством службы» вместо нецензурной лексики пишет «несомненно, эта ситуация стала следствием сложных обстоятельств развития системы и не является прямой ошибкой проектировщиков».

Малозначимым в глазах руководства может оказаться не только проект, но и обслуживание крупного продукта. Весьма показательный уровень программирования в одной социальной сети можно было оценить по пришедшему от их имени письму следующего содержания: «*Ваши фотографии были перенесены на наш новый фотохостинг. Всего было перенесено 0 фотографий*». Для того чтобы вставить в код программы рассылки проверку IF > 0, нужно, видимо, иметь не только недюжинные умственные способности, но и дополнительную квалификацию, равно как и понимание сути выполняемой задачи. С другой стороны, да и чёрт с ними, с сотнями тысяч отправленных бесполезных писем. Одной массовой рассылкой больше, одной меньше, не правда ли?

В постиндустриальной экономике сфера услуг занимает более 50 % деятельности, и эта доля растёт, например, в США она уже близка к 70 %. Представьте себе ваш бывший школьный класс, где к работе в обслуживании ориентиро-

ваны не более 25 %. Откуда же брать недостающих, да при этом еще и услужливых? Проблема, удовлетворительных решений которой на сегодняшний день не найдено. Поэтому и обсуждают введение пособий типа БОД: пусть лучше получают свой минимум и занимаются чем хотят, чем портят отношения с клиентами, мешая производительному труду остальных.

Начинающим соискателям

Для начинающих я составил небольшой словарик ключевых фраз, часто присутствующих в объявлении о вакансии. По замыслу, он должен помочь молодому соискателю вакансии программиста разобраться в ситуации и принять решение на основе более полной информации:

1. «Быстро растущая компания» – фирма наконец получила заказ на нормальные деньги. Надо срочно нанять народ, чтобы попытаться вовремя сдать работу.

2. «Гибкие (*agile*) методики» – в конторе никто не разбирается в предметной области на системном уровне. Программистам придётся «гибко», с разворотами на 180 градусов, менять свой код по мере постепенного и страшного осознания того, какую, собственно, прикладную задачу они решают.

3. «Умение работать в команде» – в бригаде никто ни за что не отвечает, документация потеряна или отсутствует с самого начала. Чтобы понять, как выполнить свою задачу, требуются объяснения коллег, как интегрироваться с уже написанным ими кодом или поправить исходник, чтобы наконец прошла компиляция модуля, от которого зависит ваш код.

4. «Умение разбираться в чужом коде» – никто толком не знает, как это работает, поскольку написавший этот код сбе-

жал, исчез или просто умер. «Умение работать в команде» не помогает, проектирование отсутствует, стандарты на кодирование, если они вообще есть, практически не выполняются. Документация датирована прошлым веком. Переписать код нельзя, потому что при наличии многих зависимостей в отсутствии системы функциональных тестов этот шаг мгновенно дестабилизирует систему.

5. «Гибкий график работы» – программировать придётся «отсюда и до обеда». А потом после обеда и до устранения всех блокирующих ошибок.

6. «Опыт работы с заказчиком» – заказчик точно не знает, чего хочет, а зачастую – неадекватен в общении. Но очень хочет заплатить по минимуму и по максимуму переложить риски на подрядчика.

7. «Отличное знание XYZ» – на собеседовании вам могут предложить тест по XYZ, где в куске спагетти-кода нужно найти ошибку или объяснить, что он делает. Это необходимо для проверки пункта 4. К собственно знанию XYZ-тест имеет очень далёкое отношение.

Тесты – особый пункт при найме. Чаще всего они касаются кодирования, то есть знания синтаксиса, семантики и «что делает эта функция».

Много лет назад по необходимости я составил небольшой сборник подобных тестов по Delphi и Transact SQL для соискателей вакансии программиста. Время от времени пользо-

вался. Частенько люди, не сумевшие ответить на большинство вопросов, просили тесты забрать с собой. Забирайте, на здоровье.

Спустя десяток лет посмотрел на те же тесты скептически. Знание технологии они ещё худо-бедно позволяют выяснить, а вот как человек мыслит – непонятно. Мой опыт говорит, что «натаскать» можно практически на любой формальный тест даже «двоечника». Поэтому не стоит мерить интеллект тестом на IQ[15]. Лучше давать испытуемому некоторый нестандартный тест, чтобы просто посмотреть на ход его мысли. Или давать один такой тест 2 раза подряд, выводя IQ не из результатов, а из прогресса верных ответов на второй итерации. Неисчерпаемым источником для неформальных тестов может послужить полная нестандартных задач книга профессионального системного программиста Чарльза Уэзерелла «Этюды для программистов» [17].

---

[15] Коэффициент интеллекта (англ. intelligence quotient).

Про CV

Cirriculum Vitae, или CV, оно же по-русски «резюме», является важной деталью вашего представления потенциальному работодателю. На Западе принято прикладывать к нему ещё и мотивационное письмо, об искусстве написания которого выпускаются целые брошюры. Поэтому ограничимся только CV.

Я сформулировал бы основные принципы хорошего резюме следующим образом:

- **Краткость – сестра таланта**. Даже в небольшой фирме ваше резюме будут просматривать несколько человек. Вполне возможно, что первым фильтром будет ассистент по кадрам, который не имеет технического образования и вообще с трудом окончил среднюю школу. Поэтому постарайтесь на первой странице поместить *всю* основную информацию: ФИО, координаты, возраст, семейное положение, мобильность, личный сайт или блог, описания своего профиля, цель соискания, основные технологии с оценкой степени владения (от «применял» до «эксперт»), образование, в том числе дополнительное, владение иностранными языками. Всё остальное поместите на 2–3 страницах.
- **Кто ясно мыслит**, **тот ясно излагает**. Все формулировки должны быть ёмкими и краткими. Не пишите «узнавал у заказчика особенности некоторых бизнес-процессов в

компании» или «разработал утилиту конвертации базы данных из старого в новый формат». Пишите «занимался постановкой задачи» или «обеспечил перенос данных в новую систему».

• **Не фантазируйте**. Проверьте резюме на смысловые нестыковки. Если на первом листе значится «эксперт по С++», но при этом в опыте работы за последние 5 лет эта аббревиатура встречается один раз в трёх описаниях проектов, то необходимо скорректировать информацию.

• **Тем более не врите**. Вряд ли кадровики будут звонить вашим предыдущим работодателям, но софтостроительный мир тесен, а чем выше квалификация и оплата труда, тем он теснее. Одного прокола будет достаточно для попадания в «чёрный список» компании, а затем через общение кадровиков и агентств по найму – ещё дальше.

• Если соискание касается технического профиля, в каждом описании опыта работы **упирайте на технологии**, если управленческого – на периметр ответственности, если аналитического – на разнообразие опыта и широту кругозора.

• **Не делайте ошибок**. Пользуйтесь хотя бы автоматической проверкой грамматики. Мало того, что ошибки производят негативное впечатление, они могут радикально изменить смысл фразы. Например, если написать «политтехнический университет»…

• **Будьте готовы**, **что далее первой страницы ваше резюме читать не станут**, а о подробностях «творческого

пути» попросят рассказать на первом собеседовании.

После обсуждения вашего сногсшибательного CV и ответного рассказа работодателя о том, как «космические корабли бороздят просторы их малого или большого театра», соискателю, как правило, следует что-нибудь спросить. Вот мой вариант. Я постарался быть краток:

Соискатель: Вы используете так называемые «гибкие» методы, например Scrum? Если да, то какова степень формализации процесса? У вас есть аналитики и проектировщики? Какие модели вы используете? Есть ли практика ежедневных утренних планёрок? Есть ли ответственные за подсистемы?

Работодатель: Да – высокая – выделенных нет – что-то рисуется в UML – обязательно! – есть, трудовой коллектив.

Соискатель: Спасибо, всего вам доброго и успехов в труде!

Про мотивацию

Мне очень нравится одна история-притча, которую приведу целиком:

> Около дома одного человека мальчишки играли в мяч: ударяли им о стены, громко кричали и смеялись. Естественно, они мешали хозяину дома. И вот в один прекрасный день он вышел к ним и сказал: «Друзья, вы так весело играете в мяч, так заразительно смеётесь и кричите, что я с удовольствием вспоминаю свое детство. Я буду платить каждому по монете, чтобы вы каждый день приходили сюда, громко кричали, смеялись и играли в мяч». Мальчишки взяли по монете и продолжили игру. На следующий день они снова пришли и получили по монете. Так продолжалось несколько дней. Но как-то хозяин подошёл к мальчишкам и сказал, что его финансовые дела не так хороши, как раньше, и он сможет платить им только по полмонеты. Он заплатил им по полмонеты и ушёл. А мальчишки поговорили и решили, что не будут стараться за полмонеты. И больше они не приходили. Так хозяин дома получил желаемые мир и спокойствие. .

Трудно сказать, почему вместо слова «стимуляция» повсеместно прижилось «мотивация». Навязли в зубах рекламные рассылки «способы мотивации персонала», кото-

рая совсем не мотивация, а стимуляция. Не иначе, консультанты хотят избежать нежелательных ассоциаций со стимулированными свинками и собаками Павлова. Или с древнеримской палкой-стимулом, при помощи которой помыкали домашним скотом.

Мотивация – исключительно внутренний механизм. Чтобы управлять им, необходимо залезать в этот самый механизм, в психику. Существуют традиционные способы: педагогика и воспитание. Они требуют многих лет, а то и смены поколений. Существуют и более быстрые варианты, связанные с химией и традиционной медициной. Наконец, есть и очень быстрые и рискованные, связанные с психотехниками, гипнозом и «зомбированием».

Поэтому, когда вам предлагают услуги по «мотивации персонала», желательно спросить, будут ли персонал бить током, колоть препаратами или ограничатся лёгким массовым сеансом гипноза.

Напротив, *стимуляция* – это внешний механизм. Он основан на выявлении мотивов и последующем их поощрении или подавлении. Стимулятор подобен катализатору для запуска химической реакции. Управление стимуляцией сводится к созданию системы стимулов, требуемых для «реакций» катализаторов. Для «реакций» – процессов работы человеческих коллективов.

Управлять мотивацией, то есть целенаправленно изменять психологию и выстраивать набор стимулов – это «две

большие разницы». Первое, по сути, требует изменения самих людей, второе – это использование имеющихся у них мотивов. Мотивировать же можно только свои поступки, но никак не образ действия окружающих.

Изгибы судьбы при поиске работы

Рассказ из реальной жизни, надеюсь, внесёт долю юмора в оказавшуюся не самой весёлой тему профессиональной ориентации.

С наступившей весной я озаботился сменой работодателя. Действительно, на дворе кризис, «троечники» сидят по своим местам, самое время поискать весёлую компанию «отличников» или, на худой конец, «хорошистов». Правда, время поиска вырастает раза в 2, но в течение месяцев так трёх-четырёх найти место вполне реально даже при финансовых запросах выше среднего. Технология ловли рыбы в мутной воде несложная, достаточно разместить резюме на одном из крупных веб-сайтов, параллельно входя напрямую в контакт с отдельными компаниями.

Как обычно, регулярно названивали мадемуазельки ранга ассистента по «человечьим ресурсам», первый их вопрос стандартен, вызван дилетантизмом в предметной области нанимаемых и потому звучит всегда: «А какую работу вы ищете, если поточнее?» То есть объясните, дяденька, что это за аббревиатуры такие у вас на первой странице CV. Если настроение хорошее, можно коротко просветить девушку, если не очень, то отвечать в стиле: «Ровно то, что написано в заголовке CV, вы его читали?»

Второй этап – выяснение, ищут ли они специалиста на конкретный проект или просто под широкий профиль. Это вопрос специфичный для самого массового работодателя – консультационно-проектных фирм. Если набор идёт «под широкий профиль», то можно вежливо начинать прощаться.

Чтобы избежать этого этапа, а также при желании найти место у конечного клиента, следует ограничить круг своего общения кадровыми агентствами или просто не указывать в резюме номер телефона, ограничившись электронной почтой.

Третий этап – отбрыкаться от приглашения на бесполезное интервью, убедив, что не стоит зря тратить время. Ведь у мадемуазелек рабочего времени навалом. Достаточно попросить прислать краткое описание требований к вакансии. В 90 % случаев это срабатывает, причём в 90 % из этих 90 % случаев требования оказываются неподходящими. В оставшихся 10 % можно соглашаться на интервью, главное – потом накануне не забыть уведомить, что по уважительной причине прийти на него нет никакой возможности.

Ладно, это все техника, которая приходит с опытом. Тем более, если времени много, например, вы сидите на пособии по безработице – это надо же так постараться в нашем расцвете лет, можно и походить по мадемуазелям, расширив круг повседневного общения.

Есть в округе довольно крупная проектная контора, назовём её «Контрабас», несколько тысяч человек, входит во французскую «десятку». Я

всячески избегаю крупных фирм общего профиля, потому что уровень технической экспертизы там весьма посредственный при ожидаемом беспорядке в управлении, с многодневным прохождением нескольких уровней бюрократии для простейших операций вроде покупки билета на скоростной поезд, буксующей из-за отсутствия названия станции назначения в корпоративной системе управления.

Как раз звонят из этой конторы. Но не ассистентка, а паренёк, и, видимо, подкованный. Так как после первой минуты сказал, что ещё через четверть часа мне перезвонит собственно начальник отдела, куда ищут работника. С руководителем мы приятно побеседовали минут 20. Выяснилось, что хотя профиль и не совсем тот, что нужен «ещё вчера», но вот очень скоро будет проект и уж там-то. . Ну и хорошо, как будет, так и созвонимся-спишемся? Спишемся.

Вечером в почтовый ящик падает анкета «Контрабаса» для соискателя на 10 (!) страницах. Я тут же ее закрыл, письмо переместил в корзину.

Ещё через пару недель позвонила не просто мадемуазелька, а уже целая мадам. Из той же конторы, но из другого подразделения. После фраз о том, как много у них вакансий по моему профилю и приглашения на интервью, пришлось отвечать, что прийти я смогу только для разговора по конкретной вакансии, а не по их множеству. Мадам несколько впала в ступор и в течение пары минут переспрашивала и объясняла, что у них вот такая процедура найма и

никак иначе нельзя. Мне было очень жаль, но раз такая процедура найма – тем хуже для найма.

Третий акт интермедии произошёл уже после того, как я нашёл себе небольшую контору человеческого размера из бывших писателей программ в школьных кружках и вышел на работу. Оказалось, что «Контрабас» с малопонятными целями умудрился купить мою новую контору.

В конце концов, по сумме обстоятельств я стал сотрудником «Контрабаса», избежав заполнения 10-страничной анкеты и нескольких раундов интервью, из которых смысл имеет только один – с непосредственным начальником или напарниками, но остальные можно не пройти по совершенно не зависящим от тебя причинам. Например, много лет назад я по неопытности пытался объяснить мадемуазельке из фирмы-посредника разницу между SQL и PL/SQL, потому что это было важно для данной вакансии. А она только улыбалась. Но по итогам выдала моему агенту заключение: «Не могу рекомендовать вашего инженера клиенту, он был со мной очень холоден. .» Я не шучу, формулировка была именно такой.

Коллеги, не будьте холодны с мадемуазельками-ассистентками из кадровых служб! Уважайте их заслуженное право на какую-то деятельность после с трудом законченной средней школы и курсов.

Технологии

Никогда не догоняйте
устремившихся вперёд.
Через пять минут, ругаясь,
Побегут они обратно,
И тогда, толпу возглавив,
Вы помчитесь впереди.

Г. Остер

Термин «гуглизация» (*googlization*) не случайно созвучен с другим, с глобализацией. И если глобализация систематично уничтожает закрытые экономики, то «гуглизация» нивелирует энциклопедические знания. Пространство практического применения эрудита сузилось до рамок игры «Что? Где? Когда?». Доступность информации снизила ее значимость, ценность стали представлять не сами сведения из статей энциклопедии, а владение технологиями.

Часть пролетариата умственного труда, способная хранить и воспроизводить технологии, превратилась в «когнитариат». Появился термин «индустриальная археология», касающийся реинжиниринга[16] систем в промышленной эксплуатации, принципы и технологии работы которых неиз-

[16] От англ. «reverse engineering» – восстановление общих проектных решений и концепций по имеющейся частной их реализации.

вестны никому из обслуживающего их персонала.

Технологии в аппаратном обеспечении, «железе», подчинены законам физики, что делает их развитие предсказуемым с достаточной долей достоверности. Зная, какие работы ведутся в лабораториях, можно предугадывать потолок их развития и предполагать сроки готовности к практическому использованию. Например, сейчас в активной фазе находятся прикладные исследования по созданию масштабируемой технологии проектирования и производства устройств, способных заменить нынешние полупроводниковые схемы. Вывод: через десятилетие мир вычислительных устройств изменится.

В противоположность этому, мир программных технологий основан на математических и лингвистических моделях и подчинён законам ведения бизнеса. Крупные капиталовложения, сделанные в существующие средства разработки, инфраструктуру и обучение пользователей, должны окупаться независимо от значения синуса в военное время, релятивистских поправок и элементной базы ЭВМ. Вывод: радикальных изменений в софтостроительной сфере ожидать не следует, ситуация находится под чутким контролем крупных корпораций и развивается эволюционно.

Тем не менее в софтостроении, даже кустарном и далёком от индустриализации, технологии составляют основу. О них мы и поговорим.

Можно ли конструировать программы как аппаратуру?

Для развития аппаратной части определяющими являются физические законы, а основой индустриализации в производстве «железа» стали проектирование и сборка устройств из стандартизованных компонентов.

Конечно, в софтостроении тоже имеются относительно стандартные подсистемы: операционные среды, базы данных, веб-серверы, программируемые терминалы и тому подобное. Однако их масштаб соответствует не компоненту в устройстве, а достаточно сложной аппаратной подсистеме вроде маршрутизатора или сервера.

Возможность собирать изделия из «кубиков» стала предметом зависти софтостроителей, вылившейся в итоге в компонентный подход к разработке. Панацеи, разумеется, не получилось, несмотря на серьёзный вклад технологии в повторное использование «кубиков», оказавшихся скорее серыми ящиками с малопонятной начинкой. Но появился целый рынок, где писатели компонентов предлагают свои изделия «компонентокидателям» – это жаргонное слово возникло в среде наиболее массового применения компонентов, где их выбирают на палитре мышкой и, протаскивая, кидают[17]

[17] Англ. Drag and drop.

на разрабатываемую экранную форму.

Для аппаратуры используется модель конечного автомата. Во-первых, она обеспечивает полноту тестирования. Во-вторых, компонент работает с заданной тактовой частотой, то есть обеспечивает на выходе сигнал за определённый интервал времени. В-третьих, внешних характеристик (состояний) у микросхемы примерно *два в степени количества «ножек»*, что на порядки меньше, чем у программных «кубиков». В-четвёртых, высокая степень стандартизации даёт возможность заменить компоненты одного производителя на другие, избежав сколько-нибудь значительных модификаций проекта.

В софтостроении использовать конечно-автоматную модель для программного компонента можно при двух основных условиях:

- Программисту не забыли объяснить эту теорию ещё в вузе (см. выше про «Круговорот»).
- Количество состояний обозримо: они, как и переходы, достаточно легко определяются и формализуются.

Второй пункт более важен. На практике количество состояний даже несложного модуля запредельно велико, поэтому программист использует их объединения в группы и применяет различные эвристики для обеспечения желаемого результата на выходе при заданном входе.

Возьмём относительно простой пример: компонент, кон-

вертирующий сумму из одной валюты в другую.

Из элементов стандартизации точно присутствуют коды валют по ISO 4217[18] и, частично, список служб, к которым компонент может обращаться (см., например, каталог служб *Financial API*). Интерфейс самого компонента не стандартизован, для возможной его замены в будущем без последующей структурной перекройки вашего приложения потребуется обернуть компонент в адаптер (привет, шаблоны!). Это поможет избежать реструктуризации при замене, но не гарантирует работоспособность на том же входном наборе.

Теперь оценим количество состояний, которые необходимо охватить для полноты модульного тестирования, раз уж мы следуем логике разработки «железа». ISO 4217 даёт список из 164 валют. Предположим, что наши входные данные:

- имеют только два знака после запятой;
- значения положительные;
- максимальная величина – 1 миллион;
- дата конвертации всегда текущая;
- мы используем только 10 валют из 164.

Несложный комбинаторный подсчёт показывает, что даже такой сильно урезанный входной набор характеризуется количеством размещений из 10 по 2, помноженным на 100 миллионов входных значений (1 миллион с шагом 0,01):

[18] Международный стандарт, регламентирующий кодификацию валют.

$10^2 \times 100\,000\,000 = 10\,000\,000\,000.$

То есть для обеспечения полноты тестирования нашего входного набора потребуется 10 миллиардов проверок! Сравните, например, с микросхемой дешифратора, преобразующего входное 4-разрядное двоичное значение в сигнал на одном из 16 выходов. Входных наборов будет всего 16, а таблица истинности состоит из $16^2 = 256$ значений.

На практике программист применит допустимую эвристику и будет тестировать, например, только несколько значений (один миллион, ноль, случайная величина из диапазона) для нескольких типовых конвертаций из 100 возможных, дополнительно проверяя допустимую точность значений на входе. При этом формальный показатель покрытия модульными тестами по-прежнему будет 100 %…

Но это ещё не всё. Микросхема работает с заданной тактовой частотой. Если, например, частота равна 1 МГц, то подав на вход набор значений, вы гарантированно через одну микросекунду получите результат на выходе.

Если же вы подадите набор значений на вход нашего компонента, то время отклика будет неопределённым. Может быть, программа отработает за секунду.

Может быть, зависнет навечно, если не предусмотрен тайм-аут. А если несколько параллельных запросов?

Поэтому вдобавок к модульному тесту необходимо программировать тест производительности (нагрузочный), ко-

торый тем не менее *не гарантирует* время отклика, а только позволяет определить его *ожидаемое значение* при некоторых условиях.

Таким образом, собрав из кучи микросхем устройство, мы уверены, что оно будет работать:

- согласно таблицам истинности;
- с заданной тактовой частотой.

Собрав же из компонентов программу, мы можем только:

- приблизительно и с некоторой вероятностью оценивать время отклика на выходе;
- в большинстве случаев ограничиться выборочным тестированием, забыв о полноте.

Если вам говорят: «Пришло время собранных из кубиков программ», будьте в курсе ограничений технологии. Очень уж далеки программные компоненты от электронных кубиков.

Безысходное программирование

Любая программа, даже созданная визуально, имеет в своей основе исходный код на каком-либо языке программирования.

Безысходное программирование – это программирование без «исходников». То есть мы пишем свой код, не имея исходных текстов используемой подпрограммы, класса, компонента и т. п.

Когда необходимо обеспечить гарантированную работу приложения, включающего в себя сторонние библиотеки или компоненты, то, не имея доступа к их исходному коду, вы остаётесь один на один с «чёрным ящиком». Даже покрыв их тестами, близкими к параноидальным, вы не сможете понять всю внутреннюю логику работы и предусмотреть адекватную реакцию системы на нестандартные ситуации. Поэтому программирование без исходников в таком сценарии превращается в настоящую безысходность и безнадёгу.

Пока цена ошибки в приложении – потеря нескольких строк введённой пользователем информации, дело может ограничиться долгоживущей записью в базе данных ошибок, закрываемой не её исправлением, а описанием обхода «граблей»[19]. Но ситуация кардинально поменяется, если цена бу-

---

[19] Грабли – синоним скрытой ошибки в программе. «Наступить на грабли» в программистском фольклоре означает выявить скрытую проблему за собствен-

дет исчисляться многими нулями потерь от упущенной сделки в торговой системе, сотнями исков клиентов, получивших неверные счета, или того хуже – аварией на производстве. Ответственность с разработчиков никто не снимал.

В рамках аудита нередко приходилось наблюдать, как правят программный код триггеров и хранимых процедур прямо в базе данных. Ассоциация с этим непотребством у меня тесно связана с утилитой debug, которая в MS DOS позволяла писать машинные команды прямо в память. Или с командой type > program.com для набора машинного кода с консоли в исполняемый файл. Понятное дело, что занимаются такими вещами при разработке программного обеспечения только от безысходности.

Частным, но частым случаем безысходного программирования является софтостроение без использования системы управления исходным кодом (*revision control system*), позволяющей архивировать и отслеживать все его изменения.

ный счёт.

Эволюция аппаратуры и скорость разработки

В 1980-х годах у японцев существовала программа по созданию ЭВМ 5-го поколения. К сожалению, цель достигнута не была, хотя проявилось множество побочных эффектов вроде всплеска интереса к искусственному интеллекту, популяризации языка Пролог, да и отрицательный опыт – тоже опыт, возможно, не менее ценный.

В итоге, спустя 20 с лишним лет, все мы – и разработчики, и пользователи – продолжаем сидеть на «числогрызах» 4-го поколения. Производительность «железа» возросла на порядки, почти упёршись в физические ограничения миниатюризации полупроводников и скорость света. Стоимость тоже на порядки, но снизилась. Увеличилась надёжность, развилась инфраструктура, особенно сетевая. Параллелизация вычислений пошла в массы на плечах многоядерных процессоров.

Прежними остались лишь принципы, заложенные ещё в 1930-х годах и названные, согласно месту, Гарвардской и Принстонской архитектурами ЭВМ. Вчерашний студент теперь пишет не на ассемблере и C, а на Java, будучи уверенным в принципиальной новизне ситуации, не всегда осознавая, что изменилось только количество герц тактовой частоты и байтов запоминающих устройств.

Возросла ли при этом скорость разработки? Вопрос достаточно сложный, даже если сузить периметр до программирования согласно постановке задачи. Тем не менее я рискнул бы утверждать, что не только не возросла, но, наоборот, снизилась.

Массовые технологии, доступные шести миллионам программистов, являются универсальными, то есть могут быть использованы для разработки большинства типов программ, пакетов и систем. Поэтому важным элементом бизнеса становится не столько сокращение срока разработки, сколько максими-зация использования стандартных сред, компонентов и фреймворков. И хотя по срокам, бюджету и количеству разработчиков владеющие специализированными технологиями выигрывают у бригады «универсалов», но возникающие при этом риски могут свести к минимуму весь выигрыш.

Конечно, специализированные средства разработки всегда обеспечат преимущества по сравнению с универсальными. Тем не менее основная разработка по-прежнему будет идти на весьма ограниченном наборе универсальных сред и фреймворков, выталкивая специализированную в ниши, где сроки и производительность являются наиболее важными.

Бывшие разработчики PowerBuilder или FoxPro неоднократно выражали мне своё недоумение по поводу того, что для простейших операций, вроде настраиваемого показа табличных наборов данных на клиенте, теперь приходится

тратить уйму времени и писать десятки строк кода, а каждая корректировка структур данных должна быть отражена во всех слоях системы. Опуская технические ограничения классических клиент-серверных приложений, нетрудно убедиться, что найти на рынке труда специалиста по PowerBuilder на порядок сложнее, чем VB.NET-программиста. К тому же поставщик среды PowerBuilder после многих перекупок за последние годы в итоге выглядит не слишком жизнеспособным.

С другой стороны, количество работающих в софтостроении женщин росло до начала 1990-х годов, после чего резко пошло на убыль. Рисунок 2 представляет ситуацию в США, но и в СССР и позднее в РФ она вряд ли отличалась.

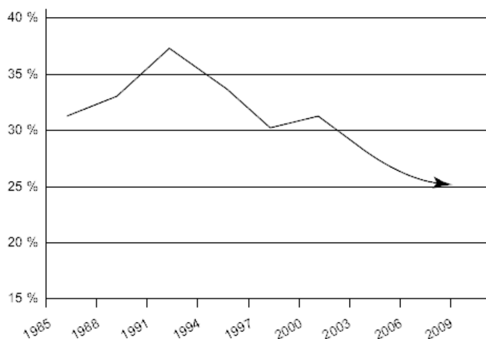


Рис. 2. Процент женщин, занятых в компьютерной отрасли в 1985–2009 годах, согласно данным американского бюро статистики труда

Такая тенденция иллюстрирует факт ухода технологий софтостроения от специализированных сред, не требующих работы на далёком от решаемой прикладной задачи уровне математических абстракций, в которых прекрасный пол почему-то считается менее способным разбираться.

В начале своей трудовой деятельности я наблюдал изображённый на графике пик в среде женщин-программистов. В отделах конструкторско-технологического центра профессия прикладного программиста прекрасно сочеталась с равноправием: мужчин и женщин среди них было примерно поровну. В разгаре был переход с больших ЭВМ на «персоналки» и локальные сети NetWare. Автоматизированные информационные системы переносили на новую платформу, используя специализированные среды разработки приложений баз данных типа FoxPro, Clipper, dBase. И женская половина коллектива успешно справлялась с поставленными перед ними задачами.

Одна умная девушка, получавшая в школе хорошие отметки по математике и без особого труда оперирующая программами и данными в среде FoxPro, после знакомства со средой C++ Builder высказалась максимально ясно: «Язык понравился, но я не поняла, зачем мне нужны эти классы…».

Впоследствии мне не раз приходилось видеть исходники программисток на вполне себе объектно-ориентированном Delphi/C++Builder. В прикладном коде никакого объектного подхода, конечно, не было, всё ограничивалось компоновкой

экранных форм стандартными элементами среды и написанием обработчиков событий в процедурном стиле.

Разумеется, это говорит не о «плохости» ООП, а о высоком уровне компетенции, необходимом, чтобы эта технология давала осязаемые преимущества. Тогда как цель прикладника – побыстрее собрать работающее решение для заказчика. Неплохим компромиссом был Visual Basic, похороненный Microsoft в начале 2000-х годов. Хотя ему и далеко до специализированных сред по удобству и скорости, VB не навязывал следование ООП, давая органично встраивать процедурную обработку между склеенными компонентами.

Про объектно-ориентированный подход мы ещё поговорим, но у меня сложилось мнение, что будучи реализованной повсеместно без малейших представлений о её применимости, эта технология сыграла не последнюю роль в вытеснении женского труда из отрасли.

Диалог о производительности

В одном из проектов у меня произошёл с заказчиком весьма характерный разговор, ярко иллюстрирующий приоритеты при освоении бюджета даже в относительно небольшой частной компании:

Заказчик: Нам необходимо рассчитать ряд показателей на основе данных одной базы, но использовать их будут из таблиц в другой базе данных.

Я: Сделаем расчёт на SQL, заполним таблицы напрямую. Базы данных у вас на одном сервере?

Заказчик: На одном, но теоретически могут быть разнесены…

Я: Значит, просто поменяется источник расчётных данных: локальный на удалённый.

Заказчик: Э-э-э… А по сравнению с пакетом сервиса интеграции[20] скорость не замедлится?

Я: Наоборот, все будет работать быстрее. Две стадии – запрос с расчётами и заливка результата – вместо трёх.

Заказчик: Ух ты, здорово! (*мнётся*)

Я (*с пониманием в голосе*): Если вы хотите привлечь к работе ещё одного человека, то мы заполним данные в расчёт-

---

[20] SSIS – SQL Server Integration Services, служба управления пакетами обработки данных, внешних по отношению к СУБД. По сути, среда для быстрой визуально-скриптовой разработки программ-конвертеров данных.

ной базе, а потом ваш сотрудник сделает пакет, который просто перекачает данные из одной базы в другую.

Заказчик (*радостно*): Да, я бы предпочёл сделать так!

Речь шла о регулярном заполнении пары таблиц объёмом примерно в сотню миллионов строк. Вместо прямого пути с настраиваемым источником данных ради приобщения к действу ещё одного «выпускника курсов» заказчик выбрал локальный расчёт с последующий перекачкой данных. В итоге используемое дисковое пространство удваивается, время увеличивается.

Служба «бизнес-интеллекта»[21] предприятия – неисчерпаемый кладезь такого рода задач для новоиспечённых специалистов курсов переквалификации и их начальников.

[21] От английского «Business Intelligence» – служба предприятия, занятая аналитической обработкой данных.

О карманных монстрах

В послужном списке одной конторы имелась небольшая и простая система ведения заказов на рекламу для книжно-журнального издательства. Полтора десятка сущностей (и таблиц), с десяток экранных форм.

Лет 10–15 назад можно было бы взять на выбор Delphi/C++ Builder, PowerBuilder, Visual Basic, FoxPro, лёгкую клиент-серверную СУБД и сделать приложение за 3–5 дней с написанием каких-то сотен строк прикладного кода. Внесение изменений типа «добавления атрибута к сущности» вместе с воссозданием инсталлятора и скрипта обновления базы данных занимало час-два.

В 2009 году приложение было сделано на платформе. NET в трёхзвенной архитектуре: сервер приложений на базе WCF, Entity Framework, СУБД SQL Server 2005 и клиент в виде подключаемого модуля (*add-in*) к Office 2007 на WinForms. Спасибо, что не на WPF. Приложение занимает примерно 20 тысяч строк на C#, из них более половины являются техническими: слой объектов доступа к данным, прокси классов для WCF и прочая начинка. Конфигурационный файл для WCF-сервера – 300 строк XML. Это больше, чем нужно написать, например, Delphi-кода для логики отображения форм во всем приложении.

Первоначальная разработка заняла у фирмы порядка трёх

недель работы одного программиста при том, что большая часть кода генерируется из модели. Отладка проблемы в канале WCF при нештатном исключении занимает часы. При добавлении атрибута изменение поднимается по всем звеньям, что также может потребовать длительного времени.

Наверное, и в 2009 году можно было бы обойтись разработкой на VB.NET приложения, напрямую работающего с СУБД через DataSet. И даже с учётом необходимости устанавливать. NET на рабочем месте, это было бы не намного хуже и медленнее, чем 10–15 лет назад.

Но механизм принятия решения базировался на других критериях:

- менеджеру, в соответствии с корпоративным стандартом, необходимо было использовать только платформы и средства Microsoft;
- программист не имел опыта разработки вне шаблонов многозвенной архитектуры и проекций объектов на реляционную СУБД, поэтому не стал рисковать.

Такие решения принимаются в мире ежечасно. Поэтому новичкам не раз предстоит столкнуться с заданием типа «быстро добавить поле в форму» и познакомиться с внутренним устройством подобных программ – карманных монстров, готовых откусить палец неосторожно сунутой руки.

ASP.NET и браузеры

Всякий раз, когда приходилось что-то делать при помощи технологии ASP. NET или просто править чей-то код, даже правильно написанный, меня не покидало ощущение копания по локоть в большой столовской кастрюле с макаронами.

Давайте вспомним историю. Успех в 1994–1995 годах первой версии бесплатной и открытой платформы PHP, называвшейся тогда Personal Home Page, показал, что веб быстрым темпом трансформируется из источника статической информации в среду динамических интерактивных приложений, доступных через «стандартный» проводник-браузер. Ниже я объясню, почему взял слово «стандартный» в кавычки. Microsoft не могла остаться в стороне и выдала собственное решение под названием ASP (Active Server Pages), работающее, разумеется, только под Windows.

Лежащий в основе названных платформ принцип был просто замечательным, хотя и совсем не новым. Логика приложений реализовывалась на стороне сервера скриптами на интерпретируемом языке, тонкий клиент-браузер в качестве терминала только отображал информацию и ограниченный набор элементов управления вроде кнопок. Вскоре выяснилось, что привыкшему к интерактивности полноценных приложений пользователю одних лишь кнопок не хватает. Тогда и в браузеры (то есть на стороне клиента) тоже включили

поддержку скриптовых языков.

В итоге исходная веб-страница, ранее содержавшая только разметку гипертекста, стала включать в себя скрипты для выполнения вначале на сервере, а затем и на клиенте. Можете представить, какова была эта «лапша» на сколько-нибудь сложной странице ASP. Многие сотни строк каши из HTML, VBScript и клиентского JavaScript.

Последующая эволюция технологии была посвящена борьбе с этой лапшой, чтобы программный код мог развиваться и поддерживаться в большем объёме и не только его непосредственными авторами. На другом фронте бои шли за отделение данных от их представления на страницах, чтобы красивую обёртку рисовали профессиональные дизайнеры-графики, не являющиеся программистами.

Однако, несмотря на значительный прогресс за последние 15 лет, производительность разработки пользовательского интерфейса для веб-приложений в разы отстаёт от автономных приложений, тех самых, что «компонентокидатели» на Visual Basic, Delphi или C++ Builder делали 15 лет назад.

Если взять простой пример отображения модального диалога, то в Delphi, Visual Basic или WinForms-приложении потребуется написать одну строку кода для вызова формы и вторую – для проверки статуса возврата. Для веб-приложения, во-первых, реализация этого сценария одними серверными скриптами невозможна, необходимо задействовать клиентские. Во-вторых, необходимо хорошо представ-

лять себе механизмы взаимодействия браузера и веб-сервера, чтобы синхронизировать вызовы и организовать передачу статуса. Наконец, веб-приложение не имеет состояния, поэтому понятие пользовательской сессии очень условное. Например, после 15-минутной паузы в деятельности клиента сервер решает, что сеанс закончен.

Теперь представьте, что под модальным окном с индикатором выполнения и единственной кнопкой «Прервать» вам надо запустить асинхронную обработку с обновлением информации в главном окне. В автономном приложении снова пишем несколько строк кода, добавляя обработчик события с делегатом из основной формы. А вот в веб… Даже краткое описание займёт несколько абзацев и будет касаться зоопарка технических ухищрений.

В качестве иллюстрации, существующая подсистема пользовательского интерфейса у одного из наших клиентов насчитывала всего около четырёх десятков экранных форм. Но для реализации только логики отображения потребовалась примерно сотня тысяч (!) строк *code-behind*[22] и Java-скриптов, несмотря на то, что создатели чётко отделили слой представлений от прикладной обработки, следовали логике «модель – представление – котроллер»[23], а общие элемен-

[22] Возможность разделения визуальной части и бизнес-логики по разным файлам. Одна из ключевых концепций в ASP.NET.

[23] От англ. «MVC, Model-View-Controller» – концепция построения приложений графического интерфейса пользователя. Развитие концепции, полностью исключающее связь модели и вида – MVP, Model-View-Presenter.

ты управления разного уровня – от собственных (*custom*) до композитных (*user*) – свели в библиотеки.

Легко проследить даже на простом примере, что для программиста помимо решения собственно прикладной задачи находится уйма забот. Основной целью такой дополнительной головной боли является платформенная независимость клиентской части приложения и максимально облегчённое развёртывание так называемого «тонкого» клиента, которым является веб-браузер.

Действительно, переносить автономное приложение между разными операционными системами и аппаратными платформами трудно. Большинство из них пишутся под Windows. Приложения на Java или WinForms.NET переносить легче, но для развёртывания требуется предустановленная среда времени исполнения (*runtime*) соответствующего фреймворка не ниже определённой версии. Гораздо меньше проблем с развёртыванием у FreePascal/Lazarus (открытый многоплатформенный аналог Delphi), новой версии Delphi XE или C++/Qt-приложений. Но, во-первых, перечисленное – далеко не самые массовые технологии, представляющие по этой причине дополнительные риски для менеджеров. Во-вторых, для обеспечения переноса и сам код, и требования к его написанию усложнятся, тогда как тестировать придётся на всех целевых платформах.

Поэтому на первый взгляд идея универсального программируемого терминала, которым является веб-браузер, под-

держивающий стандарты взаимодействия с веб-сервером, выглядит привлекательно. Никакого развёртывания, никакого администрирования на рабочем месте. Именно этот аргумент и стал решающим в конце 1990-х годов для внедрения веб в корпоративную среду. Гладко было на бумаге, но забыли про овраги…

Достаточно быстро выяснилось, что разработка приложения, корректно работающего хотя бы под двумя типами браузеров (Internet Explorer, Netscape и впоследствии Mozilla) – задача не менее сложная, чем написание кода в автономном приложении на базе переносимой оконной подсистемы (Lazarus, C++ и другие). А тестировать нужно не только под разными браузерами, но и под разными операционными системами. С учётом версий браузеров.

Поскольку отступать было поздно (см. информацию про капиталовложения в начале раздела), эту проблему решили в лоб. Корпоративная среда в отличие от общедоступного Интернета имеет свои стандарты. Поэтому при разработке веб-приложений достаточно было согласовать внутренние требования предприятия с возможностями разработчиков. К началу 2000-х годов установился фактический стандарт корпоративного веб-приложения: Internet Explorer 6 с последним пакетом обновления под Windows 2000 или Windows XP.

Под эти требования за 10 лет было написано великое множество приложений. А когда пришла пора обновлять браузеры, внезапно выяснилось, что их новые версии далеко не

всегда совместимы с находящимися в эксплуатации системами. И по этой причине простое обновление Internet Explorer 6 на 7 вызовет паралич информационных систем предприятия.

Достаточно свежий пример. В одной крупной конторе (более 10 тысяч сотрудников) система учёта рабочего времени из Internet Explorer 7, 8 или 9 на основной странице ввода зацикливалась, эмулируя скриптами щелчки мыши и подвешивая браузер. В Firefox 3 зацикливания не происходило, но не работали всплывающие окна. В более поздних версиях Firefox система не работала совсем, выдавая «*browser is not supported*». В Chrome корректно работала предварительная версия, но сданная в эксплуатацию почему-то лишилась этого качества с выдачей сообщения о несовместимости: «*The iView is not compatible with your browser, operating system, or device*».

Итого в 2011 году приложение по-прежнему стабильно работало *только в Internet Explorer 6*, выпущенном в 2001 году, то есть 10 лет назад.

За эти же 10 лет прошло огромное количество презентаций о том, что инвестиции сделаны не зря, о том, как замечательно работают веб-приложения в интранете и корпоративной среде в целом. На деле же оказалось, что, собрав свои приложения на базе веб-технологии, корпорация оказалась заложником версии и марки конкретного браузера. Даже переход с Internet Explorer 6 на 7, не говоря уже о Firefox или

Chrome, оказался катастрофой масштаба предприятия с долгими месяцами миграции и последующей стабилизации. Разумеется, если есть кому стабилизировать, ведь за 5–10 лет сменяются разработчики, уходят с рынка прежние поставщики. Для таких случаев приложение остаётся жить на виртуальной машине под старой версией операционной системы и проводника.

Предприятие оказывается один на один с веб-технологией, которая, как утверждали вначале, ничего не стоит при развёртывании и не требует администрирования на рабочем месте. Про затраты на обновление браузера, конечно, тогда никто не заикался, хотя соблюдения в необходимом объёме стандартов веб-терминалами как не было, так и нет.

Менеджеры другой фирмы стали искать решение проблемы у Google, отдав ему на откуп корпоративный документооборот, групповую работу и почту. Новое обоснование выглядело так: «Уж Google-то обеспечит совместимость приложений со своим браузером!» Не знаю, слышали ли поверившие в такой довод хоть что-нибудь об открытых системах и о печально завершившейся истории с IBM, продававшей свои большие ЭВМ вместе со своим же, привязанным к ним программным обеспечением. Человеческая история имеет свойство повторяться.

Вернемся к классическим автономным приложениям: даже в самом примитивном варианте развёртывания при использовании обычных исполняемых файлов с запуском с

разделяемого сетевого диска обновление одной программы не вызывает крах остальных. Не зря тот же SAP для работы с R/3, ключевой системой предприятия, использует самое что ни на есть полноценное оконное приложение, оставляя веб для частных случаев.

Факт наличия у большинства корпоративных клиентов только Internet Explorer версии 6 в качестве стандарта не раз оборачивался казусами. Так, обновление нашего внутреннего сервера Microsoft Exchange привело к тому, что веб-почта в Internet Explorer 6 стала работать со множеством ограничений, например, отсутствует разметка текста, нет проверки орфографии, не показывается дерево папок. Чтобы отправить почту, пришлось соединяться с сервером (!), где изначально стоял Internet Explorer 7, и работать оттуда через терминал.

Напоследок хочется пожелать коллегам, ответственным за выбор технологий, всячески обосновывать необходимость использования веб-интерфейса в вашей системе, принимая в рассмотрение другие пути.

Апплеты, Flash и Silverlight

Появившись в 1995 году, технология Java сразу пошла на штурм рабочих мест и персональных компьютеров пользователей в локальных и глобальных сетях. Наступление проводилось в двух направлениях: полноценные «настольные» (*desktop*) приложения и так называемые апплеты[24], то есть приложения, имеющие ограничения среды исполнения типа «песочница» (*sandbox*). Например, апплет не мог обращаться к дискам компьютера.

Несмотря на значительные маркетинговые усилия корпорации Sun, результаты к концу 1990-х годов оказались неутешительны: на основной платформе пользователей – персональных компьютерах – среда исполнения Java была редким гостем, сами приложения можно было сосчитать по пальцам одной руки (навскидку вспоминается только Star Office), веб-сайтов, поддерживавших апплеты, было исчезающе мало, а настойчивые просьбы с их страниц скачать и установить 20 мегабайтов исполняемого кода для просмотра информации выглядели издевательством при существовавших тогда скоростях и ограничениях трафика. Несомненно, судебная тяжба Sun в 1997 году с Microsoft, тут же прекратившей распространение Java вместе с Windows, также сыграла свою роль. Но основными объективными причинами такого

[24] От английского термина «applet» – приложеньице.

исхода были:

• универсальность и кроссплатформенность среды, обернувшаяся низким быстродействием и невыразительными средствами отображения под вполне конкретной и основной для пользователя операционной системой Windows;

• необходимость установки и обновления среды времени исполнения (Java runtime).

В 2007 году Sun утверждала, что среда исполнения Java установлена на 700 миллионах персональных компьютеров[25], правда не уточнялась её версия. В декабре 2011 года уже новый владелец – корпорация Oracle – привёл данные о том, что Java установлена на 850 миллионах персональных компьютеров и миллиардах устройств в мире[26]. Но поезд ушёл, развитие приложений на десктопах сместилось далеко в сторону по пути начинённых скриптами веб-браузеров, а рост количества мобильных устройств положил конец монополии «персоналок» в роли основного пользовательского терминала.

Тем не менее необходимость в кросс-платформенных богатых интерактивными возможностями интернет-приложе-

---

[25] «Java Runtime Environment is found on over 700 million personal computers», пресс-релиз Sun, 2007.

[26] «Java runs on more than 850 million personal computers worldwide, and on billions of devices worldwide», пресс-релиз Oracle, 2011.

ниях[27] никуда не исчезла, поскольку браузеры, нашпигованные скриптовой начинкой, обладали ещё большими техническими ограничениями и низким быстродействием даже по сравнению с апплетами. Эта ниша к началу 2000-х годов оказалась плотно занятой Flash-приложениями, специализирующимися на отображении мультимедийного содержания. Учтя ошибки Java, разработчики из Macromedia сделали инсталляцию среды исполнения максимально лёгкой в загрузке и простой в установке.

Упомянутая специализация технологии на интерактивном мультимедийном содержании веб-сайтов, включая потоковое аудио и видео, с другой стороны, оказалась непригодной для использования в разработке корпоративных приложений, продолжавшей по этой причине использовать браузеры со скриптами.

К решению проблемы подключилась Microsoft. Первым «блином» в 2005 году стала технология ClickOnce развёртывания полноценных WinForms-приложений. По-прежнему клиентское рабочее место требовало предварительно установки среды исполнения. NET версии 2. Но развёртывание и автоматическое обновление приложения и его компонентов было полностью автоматизировано. Первоначально пользователь, не имеющий прав локального администратора, устанавливал необходимую программу, просто щёлкнув по ссылке в браузере, далее запуская её с рабочего стола или из ме-

---

[27] От англ. RIA – Rich Internet Applications.

ню. Sun отреагировала молниеносно, добавив аналогичную возможность под названием Java Web Start.

Но «блин» всё-таки вышел комом. По данным AssetMetrix[28], основной парк корпоративных компьютеров в 2005 году составляли «персоналки» под управлением Windows 2000 (48 %) и Windows XP (38 %). Имея полную возможность предустановить среду. NET 2 на все эти рабочие места вместе с очередным пакетом обновлений, Microsoft не решилась на такой шаг, тем самым фактически похоронив массовое использование новой технологии разработчиками, имевшими неосторожность надеяться на помощь корпорации в развёртывании тяжёлых клиентских приложений.

Возможно, одной из причин стала потеря интереса Microsoft к WinForms, чьё развитие было заморожено, и переход в. NET 3 к более общей технологии построения пользовательских интерфейсов WPF[29], отличающейся универсальностью и большей трудоёмкостью в прикладной разработке, но позволяющей полностью разделить труд программистов и дизайнеров, что имело смысл в достаточно больших и специализированных проектах. Вот вам очередная иллюстрация к теме прогресса в производительности разработ-

---

[28] «In Q1 2005 48 % of business PCs ran Windows 2000, 38 % ran Windows XP», исследование AssetMetrix, 2005.

[29] Windows Presentation Foundation – технология Microsoft построения Windows-приложений с богатыми возможностями отображения информации и графики.

ки.

Побочным продуктом WPF стал Silverlight. По сути, это реинкарнация Java-апплетов, но в 2007 году, спустя более 10 лет, и в среде. NET. Кроме того Silverlight должен был по замыслу авторов составить конкуренцию Flash в области мультимедийных интернет-приложений.

В отличие от WPF, Silverlight вызвал больший энтузиазм разработчиков корпоративных приложений. Во-первых, для развёртывания не требовалась вся среда. NET целиком, достаточно было установить её часть, размер дистрибутива которой составлял всего порядка 5 мегабайтов. Поэтому на очередные обещания Microsoft предустановить. NET 3 можно было не полагаться, тем более при уже анонсированном. NET 3.5. Во-вторых, приложение можно было запускать не только в окне браузера, но и автономно.

Наша контора среагировала достаточно быстро, и к 2009 году в софто-строительной фабрике уже имелся номинальный генератор кода по модели для Silverlight-приложений. Ожидая взросления и стабилизации технологии, периодически подступаясь к теме, я собирал мнения коллег о встретившихся им подводных камнях.

Прежде всего насторожили меня новости про отсутствие в Silverlight отличных от юникода[30] кодировок. Их нет в кон-

[30] От англ. unicode – международный стандарт кодирования символов, позволяющий представить знаки практически всех известных алфавитов, включая иероглифы.

стантах, а Encoding.GetEncoding (1251) выдаёт ошибку. Как корректно импортировать в приложение ASCII[31]-файл? Никак. Из этого вытекала невозможность полноценной работы приложения с обыкновенным текстовым файлом данных, вроде CSV (*comma separated values*).

Прямой доступ к базам данных также отсутствовал. Можно было пойти окольными путями через COM interops и ADO, но для этого требовались очень серьёзные поводы.

И тут в корпорации, аккурат к октябрьской конференции разработчиков 2010 года, издали новый декрет: «Наша стратегия по Silverlight изменилась»[32]. Сессий по новой версии Silverlight 5 на мероприятии не было вовсе. Снова часы пробили полночь, и карета превратилась в тыкву. Приоритетом стал HTML 5.

Silverlight вырос до вполне взрослой версии 4, уже давно вышла Visual Studio 2010, где встроена поддержка разработки приложений под него. Но зададимся вопросом: «Может ли пользователь установить себе Silverlight-приложение, не будучи администратором на своем компьютере?» Ответ, мягко говоря, разочаровывающий: «Нет, не может».

Это значит, что развёртывать Silverlight-песочницы на машинах пользователей должны сами компании через своих специалистов, ответственных за инфраструктуру. Хотя в со-

---

[31] American Standard Code for Information Interchange – американская стандартная таблица кодов печатных символов.

[32] См. пресс-релиз компании Microsoft «Our strategy with Silverlight has shifted».

ответствующем официальном документе описано много способов облегчения администраторской деятельности, факт остаётся фактом: технология в своей 4-й (!) версии не может быть использована в корпоративной среде без серьёзных накладных расходов.

Итак, итог к 2012 году. Во-первых, «старые» технологии вроде автономного оконного кроссплатформенного приложения на Lazarus/FreePascal, Delphi XE или Qt/C++ по-прежнему позволяют сделать то, что нельзя сделать «новыми и прогрессивными». Во-вторых, ценность Silverlight по сравнению с полноценным. NET на уровне развёртывания практически нулевая. Видимо, по этой причине Microsoft недавно закрыла веб-сайт silverlight.net, в очередной раз оставив разработчиков в интересном положении.

Из продвигаемых Microsoft за последние 10 лет технологий для разработки полноценных пользовательских интерфейсов, не заброшенных на пыльный чердак, остался только WPF, имеющий весьма сомнительную ценность для небольших коллективов и отдельных разработчиков. WPF – это ниша крупных автономных Windows-приложений. Кроме того, сама по себе она невелика, в ней уже есть WinForms – более простой и быстрый в разработке фреймворк, к тому же переносимый под Linux/Mono. Поэтому при соответствующих ограничениях развёртывания выбор по-прежнему лежит между веб-браузером или условным Delphi, хочешь ты этого или нет…

ООП – неизменно стабильный результат

Говоря об объектно-ориентированном подходе и программировании, принято добрым словом вспоминать начало 1970-х годов и язык Smalltalk, скромно умалчивая, что понадобилось ещё почти 15 лет до начала массового применения технологии в отрасли, прежде всего, за счёт появления C++ и позднее – Объектного Паскаля. Потому что фактическим отраслевым стандартом был язык C, а Паскаль широко использовался для обучения и в основном для прикладного программирования, если не рассматривать исключения вроде первой версии Microsoft Windows. Религиозные войны 1970–80-х годов в новостных группах проходили под лозунгом «Си против Паскаля». По этой причине революционный переход сообществ на Smalltalk выглядел маловероятным, тогда как объектно-ориентированные расширения вышеупомянутых языков были восприняты положительно. Немудрено, что многие концепции Smalltalk были в них реализованы.

В начале широкой популяризации ООП, происходившей в основном за счёт языка C++, одним из главных доводов был следующий: «ООП позволяет увеличить количество кода, которое может написать и сопровождать один среднестатистический программист». Приводились даже цифры, что-

то около 15 тысяч строк в процедурно-модульном стиле[33] и порядка 25 тысяч строк на C++.

Довод в целом правильный, хотя из него совсем не следовало, что десяти программистам на C++ будет легче сопровождать общую систему, чем десяти программистам на С. Про это как-то забыли, потому что существовало много автономных проектов, управляемых процессом типа бруксовской операционной бригады[34] с главным программистом, отвечающим за всё решение. Собственно, и Бьёрн Страуструп, создатель C++, прежде всего преследовал цели увеличения производительности своего программистского труда.

Как только «главным программистом» стал «коллективный разум» муравейника, неважно мечущийся ли на планёрках «гибкой» (*agile*) разработки, прозаседавшийся ли на совещаниях по тяжёлой поступи RUP[35], проблема мгновенно всплыла, порождая Ад Паттернов[36], Чистилище нескончаемого рефакторинга[37] и модульных тестов, недосягаемый Рай

---

[33] В языке С нет понятия модуля, поэтому этот показатель несколько ниже.

[34] Ф. Брукс описывает софтостроение по принципу «операционной бригады» в своей книге «Мифический человеко-месяц»[0].

[35] Rational Unified Process – итеративная тяжеловесная методология софтостроения от компаний Rational и IBM.

[36] От англ. design pattern – шаблон проектирования.

[37] От англ. refactoring – реструктуризация и факторизация программного кода. В экстремальных методиках при отсутствии концепции системы и анализа предметной области формально требуется постоянный рефакторинг кода, при помощи которого предполагается чудесным образом прийти к хорошему решению ничего не проектируя.

генерации по моделям кода безлюдного Ада.

Термин «Ад Паттернов» может показаться вам незнакомым, поэтому я расшифрую подробнее это широко распространившееся явление:

- слепое и зачастую вынужденное следование шаблонным решениям;
- глубокие иерархии наследования реализации, интерфейсов и вложения при отсутствии даже не очень глубокого анализа предметной области;
- вынужденное использование все более сложных и многоуровневых конструкций в стиле «новый адаптер вызывает старый» по мере так называемого эволюционного развития системы;
- лоскутная[38] интеграция существующих систем и создание поверх них новых слоёв API[39].

В результате эволюционного создания Ада Паттернов основной ценностью программиста становится знание, как в данной конкретной системе реализовать даже простую новую функцию, не прибегая к многодневным археологическим раскопкам и минимизируя риски дестабилизации. Код начинает изобиловать плохо читаемыми и небезопасными

---

[38] Термин широко используется в автоматизации предприятий и происходит от «лоскутного одеяла» – разрозненного набора программ и пакетов, решающих локальные задачи подразделений.

[39] API (Application Programming Interface) – интерфейс программирования приложений, функциональность, которую предоставляет модуль, компонент или библиотека программисту.

конструкциями:

```
Services.Oragnization.ContainerProvider.ProviderInventory.I
Stacks[0].Code.Equals("S01")
```

Последствия от создания Ада Паттернов ужасны не столько невозможностью быстро разобраться в чужом коде, сколько наличием многих неявных зависимостей. Например, в рамках относительно автономного проекта мне пришлось интегрироваться с общим для нескольких групп фреймворком ради вызова единственной функции авторизации пользователя: передаёшь ей имя и пароль, в ответ «да/нет». Этот вызов повлёк за собой необходимость явного включения в. NET-приложение пяти сборок. После компиляции эти пять сборок притащили за собой ещё более 30, б€ольшая часть из которых обладала совершенно не относящимися к безопасности названиями, вроде XsltTransform. В результате объём дистрибутива для развёртывания вырос ещё на сотню мегабайтов и почти на 40 файлов. Вот тебе и вызвал функцию…

Разумеется, превратить код программы в тарелку спагетти можно без особого труда и в процедурно-модульной технологии. Разница в том, что распутывать процедурное спагетти гораздо легче, чем лапшу объектно-ориентированную. Потому что кроме вложенности вызовов процедур в ООП имеет место различного типа вложенность объектов – от наследования реализации до многоуровневых ассоциаций, и сов-

мещение в классах собственно структур данных и процедурного кода.

Несомненно, C++ является мощным инструментом программиста, хотя и с достаточно высоким порогом входа, предоставляющим практически неограниченные возможности профессионалам с потребностью технического творчества. Я видел немалое количество примеров изящных фреймворков и прочих «башен из слоновой кости», выполненных одиночками или небольшим коллективом. Но крупные проекты подвержены влиянию уже упоминавшейся гауссианы (см. рис. 1). Нормальное распределение вовлекает в процесс большое количество крепких профессионалов-середняков, которым надо сделать «чтобы работало» с наименьшими телодвижениями во время нормированного рабочего дня. Если Microsoft или Lockheed Martin – подрядчик Министерства Обороны США, имеют возможность растянуть кривую на графике вправо и вложить немалые

средства во внутреннюю стандартизацию кодирования, то в обычной ситуации оказывается, что C++, действительно увеличивавший личную продуктивность Страуструпа и его коллег, начинает тормозить производительность большого софтостроительного цеха где-нибудь в жарком субтропическом опенспейсе[40] площадью в гектар. Помимо общих проблем интеграции, на C++ достаточно просто «выстрелить се-

---

[40] От англ. openspace – большое офисное помещение, зал без перегородок с расположенными в нем рабочими местами.

бе в ногу», и человеческий фактор быстро становится ключевым риском проекта.

Если вернуться к вопросу стандартов кодирования на С ++, хорошим примером будет разработка программной начинки нового истребителя F-35 [15]. Объем разработанного кода порядка 10 миллионов строк, это даже больше, чем Windows. Следовательно, стандарт вполне пригоден к практическому использованию. Но имеются ли у вас в проекте ресурсы для того, чтобы не только обучить всех программистов 150-страничному своду правил, но и постоянно контролировать его исполнение?

Поэтому появились новые С-подобные языки: сначала Java, а чуть позже и C#. Они резко снизили порог входа за счёт увеличения безопасности программирования, ранее связанной прежде всего с ручным управлением памятью. Среды времени исполнения Java и. NET решили проблему двоичной совместимости и повторного использования компонентов системы, написанных на разных языках для различных аппаратных платформ.

Когда многие технические проблемы были решены, оказалось, что ООП очень требовательно к проектированию, так и оставшемуся сложным и недостаточно формализуемым процессом. Похожая ситуация была в середине XX века в медицине: после изобретения антибиотиков первое место по смертности перешло от инфекционных болезней к сердечно-сосудистым.

Примерно в то же время в сообществе начались дискуссии, появились первые публикации вроде уже ставшей классической «Почему объектно-ориентированное программирование провалилось?»[41]. Эксперты по ООП в своих книгах стали нехотя писать о том, что технология тем эффективнее, чем более идеален моделируемый ею мир.

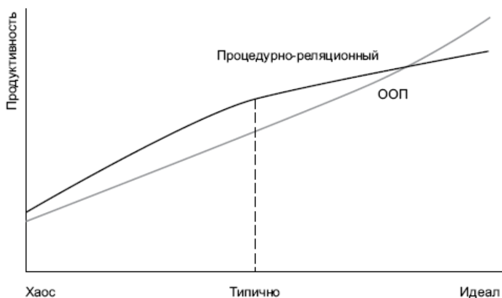


Рис. 3. Эмпирическое сравнение производительности процедурно-реляционного и объектно-ориентированного подходов в зависимости от достигнутой степени формализации моделируемого мира

Действительно, вспомним ещё раз Smalltalk. Его концепции выросли из задач построения графического интерфейса пользователя. Взглянув на любой оконный фреймворк, вы увидите искусственный мир, идеальный с точки зрения его

[41] См. публикацию «Objects Have Failed» (2000 г.) и материалы конференции OOPSLA (Object-Oriented Programming, Systems, Langauges and Applications) по данной теме в 2002 г.

авторов. Многоуровневые иерархии классов не воссозданы многолетним трудом классификации объектов окружающего мира, а выращены в виртуальных пробирках лабораторий разработчиков.

Учебники по ООП полны примеров, как легко и красиво решается задачка отображения геометрических фигур на холсте с одним абстрактным предком и виртуальной функцией показа. Но стоит применить такой подход к объектам реального мира, как возникнет необходимость во множественном наследовании от сотни разношёрстных абстрактных заготовок. Объект «книга» в приложении для библиотеки должен обладать свойствами «абстрактного печатного издания», в магазине – «абстрактного товара», в музее – «абстрактного экспоната», в редакции, типографии, в службе доставки… Можете продолжить сами.

Попытки выпутаться из этой ситуации за счёт агрегации приводят к новым дивным мирам, существующим только в воображении разработчиков. Теперь объект «книга» это контейнер для чего-то продающегося, выдаваемого, хранящегося и пылящегося. Необходимо быстро менять контекст: в магазине вкладывать в книгу товар, в библиотеке – печатное издание, в отделе «книга-почтой» – ещё какую-нибудь хреновину. Плодятся новые многоуровневые иерархии, но теперь уже не наследования (*is a*), а вложения (*is a part of*).

Изящнее выглядят интерфейсы. Но если в реальном мире книга, она и в музее – книга, то во вселенной интерфейсов

«книга в музее» – неопознанный объект, пока не реализован соответствующий интерфейс «экспонат». Дальше интерфейсы пересекаются, обобщаются, и мы получаем ту же самую иерархию наследования, от которой сбежали. Но теперь это уже иерархия, во-первых, множественная, а во-вторых, состоящая из абстрактных классов без какой-либо реализации вообще (интерфейс, по сути, есть *pure abstract class*). Если же мы отказываемся от обобщения интерфейсов, то фактически оказываемся в рамках современных реализаций модульного программирования типа Оберон[42].

Тем не менее все три подхода применимы и могут дать хороший результат при высокой квалификации проектировщиков и наличии проработанных моделей предметной области.

Одна из причин подобных злоключений в том, что концепции, выдвигаемые ООП, на самом деле не являются его особенностями за исключением наследования реализации от обобщённых предков с виртуализацией их функций. И по несчастливому стечению обстоятельств именно наследование реализации является одним из основных механизмов порождения ада наследуемых ошибок, неявных зависимостей и хрупкого дизайна. Все остальные концепты от инкапсуляции и абстракции до полиморфизма имеются в вашем распоряжении без ООП. Полиморфизм с проекциями вместо таб-

[42] Оберон – семейство языков программирования высокого уровня, разработанных Никлаусом Виртом и его школой.

лиц наличествует даже в SQL.

Мой субъективный опыт подтверждает, что за исключением фреймворков весьма абстрактного уровня, сделанных «с чистого листа» небольшими группами профессионалов высокого класса, Объектно-Ориентированный Подход на практике в большинстве случаев превращает проект или продукт, переваливший за сотню-другую тысяч строк, в упомянутый Ад Паттернов, который, несмотря на формальную архитектурную правильность и её же функциональную бессмысленность, никто без помощи авторов развивать не может.

С другой стороны, любая неясность в постановке задачи вынуждает разработчиков сосредотачиваться не на её решении, а на архитектуре, позволяющей «без особых затруднений» менять логику приложения и переходить с расчёта зарплаты колхоза на прогноз удоев фермы.

Результат неизменно стабильный…

Особо хочу остановиться на тезисе уменьшения сложности при использовании ООП для создания фреймворков. Современное состояние дел – это платформа. NET с примерно *40 тысячами* классов и типов ещё в версии 3.5. Вдумайтесь, вам предлагают для выражения потребностей прикладного программирования язык с 40 тысячами слов, без учёта глаголов и прилагательных, называя такую технологию упрощением.

Александр Сергеевич Пушкин использовал в своем твор-

честве порядка 24 тысяч слов. Толковый словарь Ожегова содержит около 70 тысяч слов. Среднестатистический русский человек использует в повседневной жизни от 5 до 10 тысяч слов[43], из них только 3 тысячи являются общеупотребительными. Получается, что даже наследник гения Пушкина способен охватить менее половины предлагаемой технологии, при том что её словарь сравним с естественным языком!

[43] Объем активного словаря образованного человека оценивается в среднем в 5–10 тысяч слов. «Сколько слов в русском языке?», Наука и жизнь, 2004, № 11.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «ЛитРес».

Прочитайте эту книгу целиком, купив полную легальную версию на ЛитРес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.