Вы инженер-программист. Что теперь?

READ ME

СУРОВЫЕ РЕАЛИИ РАЗРАБОТЧИКОВ

КРИС РИККОМИНИ
ДМИТРИЙ РЯБОЙ

ББК 32.973.2-018-02
УДК 004.415
Р50

Риккомини Крис, Рябой Дмитрий

Р50 README. Суровые реалии разработчиков. — СПб.: Питер, 2023. — 304 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1972-1

Начинающим программистам требуется нечто большее, чем навыки программирования. Столкнувшись с реальной работой, вы моментально понимаете, что самым нужным вещам, имеющим критическое значение для карьеры, не обучают ни в университетах, ни на курсах. Книга «README. Суровые реалии разработчиков» призвана восполнить этот пробел.

Познакомьтесь с важнейшими практиками инжиниринга, которым обучают разработчиков в ведущих компаниях.

Вы узнаете о том, что вас ждет при устройстве на работу, затем познакомитесь с особенностями кода промышленного уровня, эффективным тестированием, ревью кода, непрерывной интеграцией и развертыванием, созданием проектной документации и лучшими практиками архитектуры ПО. В последних главах описываются навыки гибкого планирования и даются советы по построению карьеры.

Ключевые концепции и лучшие практики для начинающих разработчиков — то, чему вас не учили в университете!

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018-02
УДК 004.415

ISBN 978-1718501836 англ.

ISBN 978-5-4461-1972-1

Оглавление

1 Предстоящее путешествие

На протяжении карьеры каждому инженеру-программисту приходится выступать в роли новичка, инженера, техлида и иногда даже руководителя. Большинство новоиспеченных инженеров обладают хорошей технической подготовкой, но не имеют практического опыта. Следующие главы помогут вам достичь первой вехи в своей карьере, то есть научиться безопасно вносить изменения в код и эффективно взаимодействовать с командой.

Достичь первой вехи не так уж и просто, поскольку нужные сведения разбросаны по Всемирной паутине или, что еще хуже, хранятся в чьей-то голове. В этой книге собрана вся ключевая информация, необходимая для достижения успеха. Но что именно подразумевается под понятием «успешный инженер-программист» и как им стать?

Пункт назначения

Для продвижения вперед по карьерной лестнице потребуется развивать свои навыки в нескольких основных направлениях.

- **Техническая подготовка.** Вы знаете основы computer science. Вы умеете использовать интегрированные среды разработки (IDE), системы сборки, отладчики и фреймворки для автоматизации тестирования. Знаете, что такое непрерывная интеграция, метрики и мониторинг,

конфигурации и системы управления пакетами. Вы активно пишете и улучшаете тестовый код. Вы учитываете особенности операций при принятии архитектурных решений.

- **Мастерство.** Вы создаете ценный продукт, решая задачи с помощью кода, и понимаете связь между вашей работой и бизнесом компании. Вам уже доводилось создавать и внедрять функции малого и среднего масштаба. Вы умеете писать, тестировать и рецензировать код. Вы готовы участвовать в дежурствах и решать операционные вопросы. Вы активны и надежны. Вы участвуете в технических совещаниях, групповых обсуждениях и собеседованиях.
- **Коммуникация.** Вы умеете четко доносить свои мысли как в письменной, так и в устной форме. Вы можете эффективно давать и получать обратную связь. В неоднозначных ситуациях способны обратиться за помощью и разъяснениями. Вы в конструктивной манере поднимаете вопросы и выявляете проблемы. Вы оказываете помощь, когда это возможно, и к вашим советам прислушиваются коллеги. Вы документируете свою работу. Вы составляете понятные проектные документы и запрашиваете обратную связь. Вы проявляете терпение и уважение в общении с другими.
- **Лидерские качества.** Вы способны самостоятельно реализовывать хорошо спланированные проекты. Вы быстро учитесь на своих ошибках. Вы хорошо адаптируетесь к нововведениям и справляетесь с нестандартными ситуациями. Вы принимаете активное участие в проектном и квартальном планировании. Вы помогаете новым членам команды осваиваться на рабочем месте. Вы предоставляете содержательную обратную связь своему руководителю.

Карта путешествия

Чтобы добраться до пункта назначения, вам потребуется карта. Материал этой главы поможет вам сориентироваться в книге и определиться с действиями в начале карьеры. Стартуете вы с пика Новичка, как и все начинающие специалисты. Оттуда вы отправитесь вниз по реке Интенсивной работы, когда приступите к написанию кода и изучению соглашений

и практик, принятых в вашей компании. Со временем вы достигнете мыса Активного участника, где внедрите ряд важных функций. По ходу вам придется преодолевать штормы в Операционном океане. В конце концов вы окажетесь в тихих водах бухты Компетентности.

Многие абзацы мы снабдили ссылками на главы. Вы можете читать все подряд или переходить к тем главам, которые вас интересуют больше всего. Некоторые ссылки на главы появляются более одного раза, и это сделано намеренно. Главы сгруппированы по темам, которые охватывают весь ваш карьерный путь. Возвращаясь к уже пройденному материалу, вы каждый раз будете открывать для себя что-то новое.

Пик Новичка

Вы начинаете свой путь в статусе новичка и знакомитесь с компанией, командой и методами работы. Посетите вводные встречи. Настройте среду разработки и доступ к системе, а также выясните особенности командной работы. Просмотрите документацию и обсудите ее с коллегами. Внесите свой вклад, заполнив обнаруженные вами пробелы в документах.

В компании может действовать программа адаптации для новых сотрудников, которая поможет вам освоиться на новом рабочем месте. В рамках такой программы вам расскажут о принятых в компании практиках, проведут экскурсию и представят руководству, а также познакомят с новыми сотрудниками из других отделов — вашими будущими коллегами. Если в компании нет такой программы, попросите своего руководителя рассказать об организационной структуре (кто за что отвечает и кто кому подчиняется), о различных отделах и их взаимосвязи. Делайте заметки.

В некоторых компаниях предусмотрены дополнительные процедуры адаптации новых инженеров-программистов, которые помогают им получить доступ к системам и настроить среду разработки, а также ознакомиться с кодом и поэкспериментировать с ним. Если в вашей компании такой процедуры не предусмотрено, вы можете ее создать! Для этого

запишите все, что вам пришлось сделать в процессе подготовки к работе. (См. главу 2 «Достижение осознанной компетентности».)

ЗАКОНЫ КАННИНГЕМА И ПАРКИНСОНА

Мы советуем вам делать заметки обо всех соглашениях, практиках онбординга и традициях команды. Будьте готовы к тому, что ваши заметки будут комментировать и исправлять. Не принимайте это близко к сердцу: ваша цель состоит не в том, чтобы написать идеальный документ, а в том, чтобы спровоцировать обсуждение, позволяющее конкретизировать детали. Как гласит *закон Каннингема*, «лучший способ получить правильный ответ в интернете — это не задать вопрос, а опубликовать неверный ответ».

Будьте готовы к тому, что дискуссия по тривиальным вопросам затянется надолго. В англоязычной литературе появился даже специальный термин bikeshed effect (дословно «эффект велосипедного сарая»), ставший метафорой закона тривиальности — его сформулировал Сирил Норткот Паркинсон. В качестве примера Паркинсон привел заседание комитета, которому поручено рассмотреть проект электростанции. Поскольку проект слишком сложен для обсуждения, комитет утверждает его за считаные минуты, а затем тратит еще около часа на выбор материалов для постройки велосипедного сарая рядом с электростанцией. Этот эффект весьма часто проявляется в технической работе.

Вам должны дать небольшое задание, чтобы вы познакомились со способами изменения кода и его внедрения в эксплуатацию. Если вы не получили такого задания, попросите дать вам возможность внести какое-нибудь небольшое, но полезное изменение. Это может быть даже обновление комментария: ваша цель — понять процедуру, а не произвести впечатление. (См. главу 2 «Достижение осознанной компетентности» и главу 8 «Доставка программного обеспечения».)

Настройте редактор кода или IDE. Используйте ту IDE, с которой работает ваша команда. Если вы не знакомы с данной средой, найдите информацию в интернете — в дальнейшем освоение IDE сэкономит вам

много времени. Настройте IDE с учетом стандартов форматирования кода, а для этого выясните, каковы они и как их применять. (См. главу 3 «Работа с кодом».)

Убедитесь в том, что руководитель включил вас в число участников всех встреч и совещаний. Напомните своему руководителю о назначении личной встречи, если это практикуется в вашей компании. (См. главу 12 «Гибкое планирование» и главу 13 «Взаимодействие с менеджментом».)

Река Интенсивной работы

Выполнив задание для новичков, вы приступите к работе над настоящим проектом вместе с командой. Скорее всего, вы будете работать с уже существующей кодовой базой. Если что-то покажется непонятным, не стесняйтесь задавать вопросы. Просите, чтобы коллеги чаще проверяли вашу работу. (См. главу 3 «Работа с кодом» и главу 7 «Ревью кода».)

На этом этапе решающее значение имеет обучение. Изучите процесс создания, тестирования и развертывания кода. Ознакомьтесь с пул-реквестами (запросами на включение изменений в коде) и ревью кода. Смело запрашивайте дополнительную информацию. Участвуйте в технических семинарах, неформальных встречах, групповых обсуждениях, программах наставничества и т. п. (См. главу 2 «Достижение осознанной компетентности», главу 5 «Управление зависимостями», главу 6 «Тестирование» и главу 8 «Доставка программного обеспечения».)

Самое время наладить отношения с непосредственным руководителем. Ознакомьтесь с его стилем работы, выясните его ожидания и поговорите о своих целях. Если ваш руководитель проводит индивидуальные встречи, будьте готовы к ним. Как правило, начальники стремятся отслеживать прогресс, поэтому спросите руководителя о предпочтительной форме отчетности о проделанной работе. (См. главу 13 «Взаимодействие с менеджментом».)

Вероятно, в этот период вы побываете на своем первом совещании по вопросам планирования, и, скорее всего, это будет собрание по

планированию спринта. Вы также можете присоединиться к ретроспективным или общим собраниям. Попросите ознакомить вас с календарным планом проекта и процессом планирования развития. (См. главу 12 «Гибкое планирование».)

Мыс Активного участника

Мыса Активного участника вы достигнете, как только начнете работать над более крупными задачами и функциями. К тому времени команда уже будет доверять вам самостоятельную работу, и вы научитесь писать удобный для оператора код промышленного уровня, позволяющий правильно управлять зависимостями и успешно проходящий тесты. (См. главу 3 «Работа с кодом», главу 4 «Написание работоспособного кода», главу 5 «Управление зависимостями» и главу 6 «Тестирование».)

На этом этапе вы уже должны помогать товарищам по команде. Участвуйте в ревью кода и будьте готовы делиться идеями и предоставлять обратную связь. Ваши коллеги могут забыть, что вы присоединились совсем недавно, поэтому смело задавайте вопросы, если чего-то не понимаете. (См. главу 2 «Достижение осознанной компетентности», главу 7 «Ревью кода» и главу 10 «Процесс технического проектирования».)

В большинстве компаний предусмотрены квартальные циклы планирования и постановки целей. Участвуйте в процессе командного планирования и определяйте цели и ключевые результаты вместе со своим руководителем. (См. главу 12 «Гибкое планирование» и главу 13 «Взаимодействие с менеджментом».)

Операционный океан

Работая над более крупными задачами, вы узнаете, каким образом код доставляется пользователям. Процесс включает в себя такие этапы, как тестирование, сборка, релиз, развертывание и внедрение. Налаживание процесса требует определенных навыков. (См. главу 8 «Доставка программного обеспечения».)

После внедрения изменений вы будете отвечать за эксплуатацию программного обеспечения вашей команды, что потребует от вас большого напряжения и выдержки: нестабильная работа ПО может вызвать недовольство клиентов. Вам предстоит отлаживать программное обеспечение в режиме реального времени, используя метрики, журналы и инструменты трассировки. На этом этапе вас также могут попросить принять участие в дежурстве. Занимаясь операционной деятельностью, вы увидите, как код отзывается на действия пользователей, и научитесь обеспечивать работу своих программ. (См. главу 4 «Написание работоспособного кода» и главу 9 «Дежурство».)

Бухта Компетентности

Теперь ваша команда уже готова поручить вам реализацию небольшого проекта. От вас потребуется написание проектной документации и помощь с планированием. В процессе разработки программного обеспечения вам откроется новый уровень сложности. Не останавливайтесь на первом варианте проекта: обдумайте возможные компромиссы и учтите развитие системы во времени. (См. главу 10 «Процесс технического проектирования», главу 11 «Создание эволюционной архитектуры» и главу 12 «Гибкое планирование».)

К этому моменту изначальный блеск вашей работы потускнеет, и вы начнете замечать недостатки в архитектуре, среде тестирования, системе сборки и развертывании. На данном этапе вы научитесь сочетать регулярную работу с обслуживанием и рефакторингом кода. Не пытайтесь все переписать. (См. главу 3 «Работа с кодом».)

У вас также будут появляться мысли относительно рабочих процессов вашей команды. Запишите свои наблюдения по поводу того, что работает, а что нет, и обсудите свои идеи в ходе личной встречи с руководителем. (См. главу 13 «Взаимодействие с менеджментом».)

Сейчас вам стоит уделить время постановке долгосрочных целей и анализу эффективности. Поработайте над этим с руководителем и запросите обратную связь от коллег. Обсудите с руководителем

свои карьерные устремления, видение дальнейшей работы, проекты и идеи. (См. главу 13 «Взаимодействие с менеджментом» и главу 14 «Навигация по карьерной лестнице».)

В путь!

Теперь у вас есть карта и вы знаете, куда направляетесь. Достигнув бухты Компетентности, вы станете полноценным инженером-программистом, способным работать вместе с командой над созданием важных функций. А наша книга поможет вам лучше ориентироваться в пути. Итак, путешествие начинается.

2 Достижение осознанной компетентности

В своей статье *Teaching for Learning* Мартин М. Бродвелл выделил четыре уровня компетентности: неосознанная некомпетентность, осознанная некомпетентность, осознанная компетентность и неосознанная компетентность. На уровне *неосознанной некомпетентности* вы не способны правильно решить задачу и не отдаете себе в этом отчета. *Осознанная некомпетентность* означает, что вы не можете правильно решить задачу, но знаете об этом. На уровне *осознанной компетентности* вы способны решить задачу с некоторым усилием. Наконец, на уровне *неосознанной компетентности* вы можете решить задачу без особого труда.

Все инженеры начинают с уровня осознанной или неосознанной некомпетентности. Даже если вы знаете все о разработке ПО (что в принципе невозможно), вам придется изучить рабочие процессы и правила конкретной компании. Вам также нужно будет освоить определенные практические навыки. Ваша цель — как можно быстрее достичь уровня осознанной компетентности.

Большая часть этой главы посвящена тому, как учиться самостоятельно и получать необходимую помощь. Учеба вне учебного заведения — особый навык, и далее вы найдете несколько советов по развитию привычки к самостоятельному обучению. Мы также подскажем, как найти баланс между слишком частым и недостаточно частым обращением за

помощью. В конце главы мы обсудим синдром самозванца и эффект Даннинга — Крюгера, которые могут вызвать у новых инженеров или неуверенность, или, наоборот, чрезмерную самоуверенность, сдерживая тем самым их рост. Мы объясним, как избегать обеих крайностей и выявлять их признаки. Самостоятельное обучение в сочетании с постановкой правильных вопросов и избеганием ловушек неуверенности и самоуверенности позволит вам в кратчайшие сроки достичь осознанной компетентности.

Научитесь учиться

Обучение поможет вам стать компетентным инженером и положит начало вашему дальнейшему процветанию. Сфера разработки программного обеспечения постоянно развивается, поэтому, кем бы вы ни были — новоиспеченным выпускником или опытным программистом, — если вы не учитесь, вы отстаете.

В этом разделе кратко описаны различные подходы к обучению. Не пытайтесь одновременно использовать их все! Это приведет лишь к выгоранию. Цените свое личное время: непрерывный рост очень важен, но не следует работать каждую свободную минуту. Выбирайте подходы, исходя из своих условий и природных склонностей.

Пройдите предварительное обучение

Первые несколько месяцев на новом месте работы потратьте на изучение рабочих процессов: это позволит вам участвовать в обсуждении вопросов разработки и решении операционных проблем, а также в дежурстве и ревью кода. На данном этапе у вас может появиться ощущение дискомфорта, поскольку вместо разработки программного обеспечения вам придется тратить время на чтение документации и знакомство с инструментами. Не волнуйтесь, для всех будет вполне ожидаемым, что вы выделите некоторое время на то, чтобы набрать обороты.

Предварительное обучение представляет собой настолько ценное вложение, что многие компании специально разрабатывают учебные программы для новых сотрудников. Например, компания Facebook проводит шестинедельный интенсивный курс обучения новых инженеров.

Учитесь на практике

Предварительное обучение не сводится к чтению документации. В ходе практической работы есть возможность узнать гораздо больше, чем в процессе чтения. Вы должны написать код и ввести его в эксплуатацию. Если вы боитесь что-нибудь сломать, не переживайте: как правило, руководители не ставят новичков в такие условия, когда те могут нанести серьезный ущерб (хотя в редких случаях новые сотрудники все же выполняют высокорискованные операции).

Постарайтесь понять, какое влияние окажет ваша работа, и действуйте с должным уровнем осторожности. Например, при написании модульного теста можно проявлять меньшую осторожность и, следовательно, действовать быстрее, чем при изменении индексов в базе данных с высоким трафиком.

ДЕНЬ, КОГДА КРИС УДАЛИЛ ВЕСЬ КОД

Во время одной из своих первых стажировок Крис работал над проектом вместе с сеньор-разработчиком. После внесения в код некоторых изменений Крису нужно было их внедрить. Сеньор-разработчик показал, как добавить код в систему управления версиями CVS. Слепо следуя инструкциям, Крис выполнил все необходимые действия, включая ветвление, тегирование и слияние. Потом вернулся к своим делам и в конце рабочего дня отправился домой.

На следующее утро Крис пришел на работу в хорошем настроении и весело поприветствовал коллег. Они попытались ответить тем же, но вид у них был удрученный. Когда Крис спросил, в чем дело, ему сообщили, что он повредил репозиторий CVS, в результате чего весь код компании был уничтожен. Его коллеги не спали ночь, отчаянно пытаясь восстановить хоть что-нибудь. В конце концов им удалось вернуть большую часть кода (за исключением коммитов Криса и некоторых других сотрудников). Крис был потрясен. Руководитель отвел его в сторону и посоветовал не переживать. Сказал, что Крис поступил правильно, работая под руководством сеньор-разработчика, однако ошибки случаются.

Подобную историю может рассказать любой программист. Старайтесь понимать, что вы делаете, но знайте, что и такие ситуации бывают.

Ошибки неизбежны. Быть инженером-программистом сложно, и все понимают, что никто не застрахован от неудач. Задача вашего руководителя и команды — создать систему защиты, способную предотвратить фатальные последствия ошибок. Допустив промах, не мучайте себя: извлеките уроки и двигайтесь дальше.

Экспериментируйте с кодом

Поэкспериментируйте, чтобы выяснить, как именно работает код: ведь и документация устаревает, и коллеги могут о чем-то забыть. Эксперименты, проведенные вне промышленной среды, совершенно безопасны и поэтому допускают применение более инвазивных техник. Например, вы знаете о вызове некоего метода, но не можете понять, как именно он осуществляется. Проведите эксперимент: сгенерируйте исключение и отобразите трассировку стека или подключите отладчик для определения пути вызова.

Отладчики — ваши лучшие друзья при проведении экспериментов с кодом. Вы можете использовать их для приостановки выполнения кода, а также для просмотра запущенных потоков и для трассировки стека и значений переменных. Подключите отладчик, инициируйте событие и осуществите пошаговое выполнение кода, чтобы увидеть, как именно код обрабатывает данное событие.

Несмотря на то что отладчики являются мощными инструментами, иногда понять поведение программы проще всего с помощью нескольких строк журнала или операторов `print()`. Вы наверняка знакомы с этим методом, однако имейте в виду, что в сложных ситуациях, особенно в случае с многопоточными приложениями, отладка с помощью операторов `print()` может приводить к путанице: операционные системы будут буферизовать записи в стандартный вывод, задерживая отображение данных в консоли, а сообщения, выводимые несколькими потоками, делающими запись в стандартный вывод, будут чередоваться.

Один простой, но на удивление эффективный метод заключается в добавлении оператора `print()` в начале выполнения программы. Такое действие позволит легко отличить модифицированную версию программы от исходной, и вам не придется тратить много часов на понимание загадочного поведения программы, если вместо ее измененной версии вы запустите исходную.

Читайте

Выделите часть рабочего времени на чтение. Может быть немало различных источников: документы, составленные вашей командой, проектная документация, код, бэклоги, книги, статьи и технические сайты. Не пытайтесь прочитать все сразу. Начните с документов команды и проектной документации, чтобы получить общее представление о том, как устроен проект. Уделите особое внимание обсуждениям компромиссов и контекста. После чего можете сосредоточиться на подсистемах, имеющих отношение к вашим первым полученным заданиям.

Как говорит Рон Джеффрис, «код никогда не лжет; порой обманывают комментарии» (https://ronjeffries.com/articles/020-invaders-70ff/i-77/). Читайте код, поскольку он не всегда точно соответствует проектной документации! Не ограничивайтесь кодовой базой своей компании — обращайтесь к высококачественному открытому исходному коду, в частности коду библиотек, с которыми работаете. Не читайте код от начала до конца, как роман: используйте IDE для навигации по кодовой базе. Создайте граф потока управления и состояний для ключевых операций. Изучите структуры данных и алгоритмы. Обратите внимание на обработку пограничных случаев. Следите за идиомами и стилем, выучите местный сленг.

Незавершенные задачи или проблемы отслеживаются с помощью *тикетов*. Ознакомьтесь с тикетами своей команды, чтобы понять, над чем работают ваши коллеги и что вас ожидает в ближайшее время. Хорошим местом для поиска тикетов является бэклог. Старые задачи делятся на три категории: более не актуальные; полезные, но второстепенные; важные, но слишком сложные, чтобы их можно было решить в данный момент. Определите, к какой категории относятся тикеты вашей команды.

Печатные издания и онлайн-ресурсы дополняют друг друга. Книги и статьи отлично подходят для более глубокого изучения предмета, однако следует иметь в виду, что содержащиеся в них сведения, несмотря на свою надежность, могут быть устаревшими. Напротив, такие онлайн-ресурсы, как сообщения в Twitter, блоги и информационные рассылки, содержат менее надежные сведения, но более подходящие для отслеживания текущих тенденций. Только не спешите реализовывать последние идеи, почерпнутые на сайте Hacker News: иногда быть скучным совсем неплохо (подробнее об этом мы поговорим в главе 3).

Присоединитесь к группе для совместного чтения, чтобы быть в курсе последних исследований. Выясните, организует ли ваша компания подобные группы. Если нет, подумайте о том, чтобы организовать ее самостоятельно. Вы также можете присоединиться к местному клубу сообщества Papers We Love, участники которого регулярно читают и обсуждают научные статьи по компьютерной тематике.

УЧИТЕСЬ ЧИТАТЬ КОД

В начале карьеры Дмитрию поручили разобраться в унаследованном Java-приложении. Руководитель хотел внести в него некоторые изменения, а Дмитрий был единственным человеком в команде, кто чувствовал себя достаточно комфортно при работе с языком Java. Исходный код был полон, скажем так, особенностей. В качестве имен переменных использовались буквы *a*, *b*, *c* и т. д. Что еще хуже, переменная *a* в одной функции превращалась в переменную *d* в другой. Не было ни истории изменений кода, ни тестов, а разработчик приложения давно покинул компанию. В общем, настоящее минное поле.

Всякий раз, когда нужно было что-то изменить в кодовой базе, Дмитрий полностью погружался в код и внимательно его читал, переименовывал переменные, отслеживал логику, набрасывал схемы на бумаге, экспериментировал. Работа шла очень медленно. Однако со временем он оценил кодовую базу, которая, как выяснилось, выполняла весьма сложные операции. Дмитрия восхищала способность разработчика программы удерживать все это в голове без нормальных имен переменных. Однажды за обедом он выразил свое восхищение, и коллега посмотрел на него так, будто у Дмитрия выросла вторая голова: «Но у нас нет оригинального исходного кода. Вы работаете с выводом декомпилятора. Никто в здравом уме не стал бы писать код подобным образом!»

Мы не рекомендуем учиться читать код именно так, однако стоит отметить, что данный опыт научил Дмитрия не спешить, вникать в прочитанное и никогда не полагаться на имена переменных.

Смотрите презентации

Хорошая презентация может многому научить. Начните с просмотра видеопрезентаций, записанных в прошлом: это могут быть как записи мероприятий вашей компании, так и внекорпоративные видеоролики на YouTube. Смотрите обучающие программы, лекции, посвященные техническим вопросам, и презентации, проводимые в рамках конференций. Попросите коллег порекомендовать вам хороший контент. Для экономии времени можно увеличивать скорость воспроизведения видео в 1,5 или даже 2 раза. Однако не смотрите пассивно: делайте заметки, чтобы не забыть важные моменты, и уточняйте любые незнакомые понятия или термины.

Участвуйте в неформальных встречах типа *brown bag* (короткая презентация или семинар) и технических обсуждениях, если ваша компания их организует: они проводятся прямо на рабочем месте, поэтому вам не придется тратить время на дорогу. На подобных мероприятиях, как правило, обсуждаются рабочие вопросы вашей компании, что позволяет получить по-настоящему актуальную информацию.

Посещайте митапы и конференции (иногда)

Конференции и митапы хороши для нетворкинга и поиска новых идей. Их стоит посещать время от времени, однако не переусердствуйте. Отношение «сигнал/шум», то есть отношение релевантного контента ко всему контенту в целом, часто бывает низким, а кроме того, многие презентации впоследствии публикуются во Всемирной паутине.

Существуют три типа конференций: научные конференции, собрания по интересам и выставки-презентации, устраиваемые поставщиками. Научные конференции отличаются замечательным контентом, однако их посещение лучше заменить чтением научных статей и участием пусть в менее крупных, зато целенаправленных обсуждениях. Встречи по интересам отлично подходят для получения практических советов и общения с более опытными специалистами: поучаствуйте хотя бы в нескольких таких встречах. Выставки-презентации, организуемые

поставщиками, — самые масштабные и значимые. Они представляют собой маркетинговый инструмент для технологических компаний и не подходят для обучения. На них можно хорошо провести время с коллегами, однако стоит ограничиться одним подобным мероприятием в год. Опять же, попросите коллег порекомендовать вам самые лучшие выставки. Имейте в виду, что некоторые работодатели оплачивают и билеты на такие мероприятия, и проезд с проживанием.

ПОДДЕРЖИВАЙТЕ СВЯЗИ С УНИВЕРСИТЕТАМИ

Несколько лет назад Дмитрий и его коллега Питер Альваро изо всех сил пытались заставить работать свое хранилище данных и думали, что правильным решением было бы распределение задач агрегации по узлам кластера серверов. В процессе работы Питер обнаружил документ с описанием модели MapReduce, выпущенный чуть ранее компанией Google. Оказалось, что Google уже делала именно то, что предлагали Питер и Дмитрий! Затем коллеги нашли другие интересные статьи и решили поискать людей, с которыми можно было бы их обсудить. Дмитрий выяснил, что в Калифорнийском университете в Беркли регулярно проводятся открытые встречи для всех желающих, посвященные обсуждению вопросов, связанных с базами данных. Питер и Дмитрий стали их завсегдатаями (но к бесплатной пицце не притрагивались до тех пор, пока свою долю не получали студенты). Иногда они даже принимали участие в разговоре!

Процесс их обучения перешел на новый уровень. В конце концов Питер остался в университете Беркли и поступил в докторантуру: теперь он профессор Калифорнийского университета в Санта-Крузе. Дмитрий тоже поступил в аспирантуру. Спустя два года после их ухода дорогостоящее хранилище данных заменили распределенной системой агрегации с открытым исходным кодом Hadoop.

Если вы почувствуете, что перестали учиться, сходите в местный университет. Как правило, там есть масса программ, открытых для публики. Расширьте свой круг общения. Поступать в аспирантуру при этом необязательно.

Попробуйте шедоуинг и парное программирование

Шедоуинг предполагает наблюдение за тем, как более опытный сотрудник выполняет то или иное задание. Наблюдатель в данном случае является активным участником: он делает записи и задает вопросы. Шедоуинг — отличный способ освоить новый навык. Чтобы получить максимальную пользу, выделите время до и после сессии для планирования и подведения итогов.

Когда будете готовы, поменяйтесь ролями. Пусть теперь сеньор-разработчик наблюдает за вами. При этом он тоже должен предоставить обратную связь. Кроме того, он сможет подстраховать вас, если что-то пойдет не так. Это хороший способ подготовиться к таким пугающим мероприятиям, как собеседование.

Парное программирование также отличный метод обучения: два инженера пишут код вместе, вводя его по очереди. К подобному нужно привыкнуть, но это один из самых быстрых способов обмена знаниями. Сторонники парного программирования утверждают, что таким образом улучшается качество кода. Если ваши коллеги не возражают, мы настоятельно рекомендуем попробовать данный метод. Парное программирование приносит пользу не только младшим инженерам — извлечь из него выгоду могут сотрудники всех уровней.

Некоторые компании поощряют наблюдение за работой и других специалистов. Например, наблюдение за работой сотрудников отдела поддержки клиентов и отдела продаж — отличный способ больше узнать о клиентах компании. Запишите свои наблюдения и поделитесь ими с коллегами. Поработайте с руководителем и сеньор-разработчиками над приоритизацией идей, вдохновленных этим опытом.

Экспериментируйте со сторонними проектами

Работа над сторонними проектами позволит познакомиться с новыми технологиями и идеями. Вы можете пропустить все, что связано с разработкой программного обеспечения, то есть тестирование, операционную работу, проверку кода и т. д.: игнорируя эти процессы, можно быстро