Министерство науки и высшего образования
Российской Федерации
Ярославский государственный университет им. П. Г. Демидова
Кафедра компьютерных сетей

И. В. Парамонов, А. М. Васильев

Социально-коммуникативные аспекты промышленной разработки программного обеспечения

Учебно-методическое пособие

Ярославль
ЯрГУ
2020

УДК 331.543:004.41(075.8)
ББК 3 973.2-018.2я73
П 18

Рекомендовано
Редакционно-издательским советом университета
в качестве учебного издания. План 2020 года.
Рецензент
кафедра компьютерных сетей Ярославского
государственного университета им. П. Г. Демидова.

Парамонов, Илья Вячеславович.

П 18 Социально-коммуникативные аспекты промышленной разработки программного обеспечения : учебно-методическое пособие / И. В. Парамонов, А. М. Васильев ; Яросл. гос. ун-т им. П. Г. Демидова. — Ярославль : ЯрГУ, 2020. — 40 с.

Учебно-методическое пособие освещает круг вопросов, связанных с деятельностью специалистов в ИТ-компаниях и эффективной коммуникацией между ними. Оно включает в себя обзор принципов функционирования ИТ-бизнеса, деятельности разработчиков и менеджеров, а также способов построения их карьеры. В конце каждой главы предлагается набор вопросов для обсуждения на семинарах.

Предназначено для студентов, обучающихся по дисциплине «Социально-управленческие аспекты промышленной разработки».

Библиогр.: 19 назв.

УДК 331.543:004.41(075.8)
ББК 3 973.2-018.2я73

Учебное издание
Парамонов Илья Вячеславович
Васильев Андрей Михайлович
Социально-коммуникативные аспекты промышленной разработки программного обеспечения
Учебно-методическое пособие
Редактор, корректор Л. Н. Селиванова
Компьютерный набор, вёрстка И. В. Парамонов
Подписано в печать 30.09.2020 г. Формат 60×84/16.
Усл. печ. л. 2,3. Уч.-изд. л. 2,0. Тираж 4 экз. Заказ
Оригинал-макет подготовлен в редакционно-издательском
отделе Ярославского государственного университета.
Ярославский государственный университет
150003, Ярославль, ул. Советская, 14.

Оглавление

Введение **4**

1. Разработка программного обеспечения: организация бизнеса и роли сотрудников **5**
1.1. Типичные схемы организации ИТ-бизнеса 5
1.2. Основные роли сотрудников, вовлечённых в процесс разработки программного обеспечения 7
1.3. Вопросы для обсуждения на семинарах 10

2. Деятельность разработчика в ИТ-компании **11**
2.1. Постановка задач 12
2.2. Типичные задачи проектирования, разработки и сопровождения исходного кода программного обеспечения 13
2.3. Обсуждение вопросов с коллегами 15
2.4. Методы и техники организации работы разработчика 18
2.5. Вопросы для обсуждения на семинарах 20

3. Деятельность менеджера в ИТ-компании **22**
3.1. Менеджер и его функции 22
3.2. Типы менеджеров в модели PAEI 24
3.3. Взаимопонимание менеджера и разработчика 25
3.4. Лидерство менеджера 26
3.5. Вопросы для обсуждения на семинарах 28

4. Карьерный путь разработчика **29**
4.1. Варианты самореализации в ИТ-индустрии 29
4.2. Возможные пути развития специалиста в иерархии компании и по ступеням профессионального мастерства 31
4.3. Интересы компании и сотрудника, их взаимная ценность друг для друга 34
4.4. Процесс превращения разработчика в менеджера . . 36
4.5. Вопросы для обсуждения на семинарах 38

Список литературы **38**

Введение

В современном мире разработка программного обеспечения давно стала деятельностью, осуществляемой большими коллективами специалистов различных специальностей. Успех этой деятельности напрямую зависит от эффективности взаимодействия всех вовлечённых участников. К сожалению, окончившие университет специалисты зачастую не вполне осознают те требования, которые накладывает на них современная индустрия информационных технологий (ИТ). Это серьёзно осложняет выполнение профессиональных обязанностей на рабочем месте и затрудняет карьерный рост.

Данное пособие осуществляет введение в круг вопросов, связанных с деятельностью специалистов в ИТ-компаниях и эффективной коммуникацией между ними. Обзор принципов функционирования ИТ-бизнеса, деятельности разработчиков и менеджеров, а также способов построения их карьеры призван смягчить профессиональную социализацию молодых специалистов и обеспечить успешное саморазвитие после выпуска из вуза.

Учебно-методическое пособие предназначено для студентов магистратуры по направлениям «Прикладная математика и информатика», «Фундаментальная информатика и информационные технологии», «Прикладная информатика (в экономике)». При обучении в магистратуре эти студенты часто уже обладают некоторым опытом трудовой деятельности по специальности, что должно упростить для них восприятие изложенного материала. Размещённые в конце каждой главы вопросы для обсуждения на семинарах и ссылки на дополнительную литературу (в основном в виде статей, написанных специалистами ИТ-индустрии) должны способствовать его закреплению. Кроме того, первоначальное ознакомление с пособием может быть полезно студентам-младшекурсникам в качестве введения в специальность.

1. Разработка программного обеспечения: организация бизнеса и роли сотрудников

1.1. Типичные схемы организации ИТ-бизнеса

В рамках индустрии информационных технологий существует множество схем построения бизнеса. Рассмотрим наиболее распространённые в настоящий момент.

Создание собственного продукта. В данной схеме компания самостоятельно определяет требования к продукту, порядок и способ их реализации. Для успешного функционирования согласно этой схеме компания должна обладать как техническими компетенциями в разработке программного обеспечения (ПО), так и хорошим пониманием предметной области. Если любое из этих требований не будет удовлетворено, шансы на успешное завершение проекта значительно уменьшатся. Также важно обладать достаточным бюджетом для разработки, так как вся деятельность финансируется компанией самостоятельно.

Продукт может разрабатываться либо для внутреннего потребления, либо для продажи в виде типового решения. В первом случае прибыль получается в результате оптимизации затрат на другие работы или за счёт организации новых видов работ на основании созданного ПО. Компании начинают разработку собственных продуктов, если существующие не подходят для решения их задач или если риски, связанные с прекращением поддержки внешним разработчиком, неприемлемы.

В случае создания типового решения компания получает прибыль за счёт продажи копий продукта конечным потребителям и за счёт оказания услуг по поддержке разработанного ПО. Одним из важных рисков при ведении бизнеса становится конкуренция с другими решениями в выбранной области, причём её необходимо будет вести как на технологическом, так и на маркетинговом уровне на протяжении всего жизненного цикла продукта.

Разработка продукта на заказ. В данной схеме требования к программному обеспечению определяет заказчик, а компания-исполнитель согласует с ним порядок и способ реализации. Исполнитель ответственен за качественную и точную по срокам техническую реализацию решения, а заказчик берёт на себя большинство

остальных рисков, включая маркетинг решения и финансирование. Для реализации данного подхода компании-исполнителю необходимо иметь компетенции в сфере разработки и сопровождения ПО.

Создаваемый продукт может являться доработкой существующего решения или совершенно новой разработкой. В первом случае компания-разработчик должна иметь опыт в работе с целевой технологической платформой. Примерами такого подхода являются разработка веб-сайтов с использованием распространённой технологии или создание конфигурации для системы управления контентом (CMS). Некоторая унификация базы решения позволяет заказчику иметь возможность замены компании-исполнителя.

При создании нового продукта заказчик и исполнитель перед началом работ должны удостовериться, что не существует подходящих типовых решений или что их эксплуатация будет слишком затратной. Например, к первой группе относятся проекты для работы с физическими устройствами или на специализированных программных платформах. Ко второй — проекты по созданию вебсистем для автоматизации внутренних процессов компаний, например больших сетевых магазинов или производств.

Аутсорсинг можно рассматривать как отдельную схему, так и в качестве разновидности предыдущей. Основное отличие от разработки продукта на заказ состоит в том, что заказчик делегирует компании-исполнителю не создание проекта целиком, а конкретную задачу или выделенный процесс с чётко определёнными условиями. На аутсорсинг может быть передана реализация любой части приложения, но зачастую заказчики ограничиваются второстепенными функциями. Примеры передаваемых процессов — системное тестирование новых выпусков на всех целевых устройствах, взаимодействие с пользователями через социальные сети, системное администрирование.

Аутстаффинг. В данной схеме организации бизнеса заказчику передаются отдельные специалисты или целая команда для выполнения задач без оформления в штат компании. Одна из причин применения аутстаффинга — стремление заказчика оптимизировать издержки компании и снизить риски, связанные с решением трудовых споров. Другая вероятная причина — необходимость оперативного расширения штата компании. Например, заказчику требуется перенести уже разработанное и успешно продаваемое

Android-приложение на iOS, поэтому ему нужна опытная команда разработки в этой области.

Компания-исполнитель получает прибыль от передачи своих сотрудников для решения технических задач компании-заказчика. Ключевыми процессами для исполнителя становятся поиск, подготовка кадров и управление ими, а оставшиеся процессы реализуются заказчиком.

ИТ-консалтинг подразумевает консультирование заказчика, направленное на информационную поддержку бизнес-процессов и позволяющее получить независимую экспертную оценку проекта. Продуктом деятельности являются экспертное заключение, рекомендации по изменению внутренних процессов заказчика, помощь по их внедрению. В отличие от предыдущих схем, эксперты привлекаются на более короткие сроки при наличии серьёзной необходимости, зачастую не систематически.

Ключевой задачей компании, работающей по данной схеме, становится поиск квалифицированных кадров, способных выполнять задачи системного анализа, и правильное позиционирование на рынке услуг по консультированию. Зачастую компании такого рода демонстрируют активную медийную активность, ведут один или два продукта, на которых демонстрируют качество своей экспертизы в выбранных областях специализации. Например, компания может разрабатывать веб-приложение, выкладывать записи в блоге о реализации его различных подсистем и особенностях использования технологий, а прибыль получать за счёт консультирования других компаний по применению данных технологий.

1.2. Основные роли сотрудников, вовлечённых в процесс разработки программного обеспечения

В небольших компаниях задачи между сотрудниками обычно распределяются в зависимости от их компетенций, а также текущего уровня занятости. В больших компаниях для облегчения управления вводятся должности с жёстко фиксированным списком обязанностей и сферой ответственности. В данном разделе рассмотрим роли сотрудников компании, занимающейся созданием собствен-

ного продукта, поскольку в это случае в процессе разработки будет задействовано максимальное количество специальностей.

Бизнес-аналитик является связующим звеном между конечными пользователями и разработчиками. Он должен хорошо разбираться в предметной области, эффективно идентифицировать проблемы клиентов, за решение которых они готовы заплатить. Работа специалиста ведётся непрерывно: анализ проводится при создании продукта с нуля, при добавлении новой функции, при выходе нового конкурента на рынок, при изменении законодательства и так далее.

Аналитику необходимо иметь представление о структуре проекта в целом, направлениях его развития и возможностях команды разработчиков. Это позволяет формировать предложения, которые будут одновременно полезны пользователям и которые могут быть реализованы за разумное время. Специалист-аналитик формализует требования к задачам в спецификации, то есть переводит описание проблемы с языка предметной области на язык разработки ПО с учётом всех ключевых деталей реализации.

Разработчик занят созданием продукта, решающего задачи пользователя. На низком уровне в его обязанности входит написание корректного исходного кода приложения в соответствии со спецификацией и автоматических модульных тестов. Ключевыми приоритетами для данной деятельности являются точность и скорость реализации функций, а также минимизация количества ошибок.

На высоком уровне разработчику необходимо также заботиться об архитектуре приложения, чтобы обеспечить возможность добавления новых функций в дальнейшем. Он также может формировать предложения по новой функциональности, отталкиваясь от технических возможностей ПО и собственного понимания предметной области. Чем больше реализованных разработчиками функций приложения использует пользователь, тем труднее ему будет перейти на конкурирующие решения и тем большую прибыль может получить компания.

Дизайнер проектирует внешний вид пользовательского интерфейса и формирует механику взаимодействия с пользователем (в английском языке используется термин user experience — пользовательский опыт). При создании интерфейса специалист в первую очередь акцентирует внимание на внешнем виде: понятность рас-

положения элементов управления, приятность цветовой гаммы, доступность текста. При разработке механики взаимодействия с пользователем акцент делается на логике работы с приложением: насколько удобно и просто пользователь может достичь своих целей. Понятные интерфейсы позволяют пользователям самостоятельно разбираться в функциях приложения с минимальным привлечением документации и обучающих материалов.

Дизайнеры передают разработчикам макеты пользовательского интерфейса и сценарии их использования. Если для приложения внешний вид является ключевой характеристикой, например, это так для многих веб-приложений, тогда макеты создаются цветными и полноразмерными под каждое целевое разрешение экрана. Для большинства бизнес-приложений достаточно схематичных макетов.

Тестировщик ответственен за обеспечение качества приложения. Этот специалист выполняет систематическое тестирование приложения, как правило, на системном уровне, в ходе которого проверяет, что новые и старые функции работают согласно спецификации на всех целевых платформах. При обнаружении проблемы тестировщик формирует детальное описание и отправляет его ответственному лицу (менеджеру или разработчику).

Ошибки в продукте могут возникнуть в результате работы специалистов любого уровня: аналитик мог предложить интегрировать функциональность в неправильный модуль, например поместить функции для кассира в модуль управления; разработчик мог проверить работу функции только под одной версией веб-браузера; дизайнер мог предложить интерфейс, рассчитанный лишь под большие экраны. Если проблемы будут систематически выявляться не компанией-разработчиком, а конечными пользователями, то они будут испытывать недовольство и искать способы замены такого приложения аналогами.

Технический писатель документирует техническую информацию о продукте для её использования как внутри компании, так и за её пределами.

Внутренняя документация обычно описывает выработанные архитектурные решения и программные интерфейсы, тем самым обеспечивая возможность разработки больших проектов силами множества разработчиков. При отсутствии такой документации коман-

де разработчиков остаётся полагаться только на передачу знаний об устройстве продукте в частном порядке или самостоятельное изучение исходного кода, что обычно неэффективно.

Внешняя документация включает в себя руководства по использованию и администрированию продуктов. Благодаря ей пользователи ПО могут самостоятельно разбираться в деталях работы требующихся функций. Наличие и качество документации зачастую является существенным фактором для приобретения приложения корпоративными заказчиками. Кроме того, специалистам технической поддержки проще общаться с клиентами и решать возникшие у них вопросы, если существует хорошая пользовательская документация.

Системный администратор ответственен за обеспечение бесперебойного доступа сотрудников компании и её клиентов как к физическим устройствам, так и к информационным ресурсам. Если разработчики не могут выполнять свои задачи из-за поломки компьютера, время доставки функционала увеличивается, а прибыль компании убывает. Если же пользователи регулярно не могут воспользоваться приложением или частью его функциональности, то они будут искать более надёжные аналоги. Таким образом, администратору необходимо обеспечивать бесперебойную работу всей инфраструктуры и внедрять средства предотвращения простоев, чтобы минимизировать время на восстановление.

Менеджер занимается в первую очередь организацией самого процесса создания продукта. В его задачи входит распределение задач между планируемыми выпусками, назначение их работникам, мониторинг, анализ рисков и, самое главное, решение любых проблем во взаимодействии сотрудников между собой и клиентами. От быстроты и качества разрешения таких проблемных ситуаций может зависеть исход всего проекта.

1.3. Вопросы для обсуждения на семинарах

1. Какие ключевые риски стоят перед компанией в зависимости от схемы организации её бизнеса?
2. Может ли одна компания использовать сразу несколько схем организации бизнеса? Почему?

3. Предположите, какие схемы организации бизнеса требуют вложения в продвижение своих товаров или услуг. На какую целевую аудиторию будут направлены данные действия?
4. Предположите, как размер компании влияет на распределение ролей сотрудников и задач между ними.
5. Стоит ли одному работнику совмещать различные роли? Совмещение каких ролей будет положительно сказываться на его деятельности, а каких — нет?
6. Должен ли разработчик разбираться в предметной области приложения или ему достаточно ориентироваться на спецификации, предоставляемые аналитиками?
7. Должен ли тестировщик заниматься проверкой исходного кода приложения? Должен ли он программировать автоматические тесты?
8. Должен ли разработчик заниматься тестированием добавленной им функциональности на системном уровне?
9. Определите, какие должности, рассмотренные в этой главе, могут соответствовать ролям, определяемым в методологии Scrum [7].
10. Существует мнение, что команда разработки должна самостоятельно решать все задачи, причём без деления на конкретные должности. Обсудите положительные и отрицательные стороны такой организации процесса разработки.

Для подготовки к семинарам рекомендуется использовать следующую литературу: [5; 7; 11; 17].

2. Деятельность разработчика в ИТ-компании

После получения образования по специальности у многих программистов складывается впечатление, что разработка ПО — это только работа над исходным кодом, включающая написание эффективных алгоритмов для обработки данных и автоматизированных тестов. Данный взгляд на разработку поддерживается и при первоначальном трудоустройстве: от начинающих разработчиков требуют решать именно такие задачи, предоставляя достаточно точные спецификации. Однако исключительно навыков решения этих

задач недостаточно, чтобы клиенты были удовлетворены разработанным ПО. В связи с этим имеет смысл рассмотреть различные аспекты деятельности разработчика более подробно.

2.1. Постановка задач

В основе создания качественной функциональности лежит детальное понимание проблемы, с которой сталкивается пользователь, и адекватность подхода, выбранного для её решения. Перед тем, как приступать к написанию исходного кода, разработчик должен получить как можно больше информации о задаче по реализации соответствующей функциональности:

— кто конкретно является пользователем;
— какая проблема данного пользователя решается;
— какие существуют требования по производительности;
— каковы сроки реализации задачи и выделенный временной бюджет;
— каков приоритет задачи;
— зависят ли другие задачи от её решения.

Так следует поступать и начинающим разработчикам, и опытным, поскольку не существует наилучших общих решений и требования будут варьироваться в зависимости от конкретных задач и проектов.

Перед началом реализации задачи разработчику следует составить детальное представление о способе её решения и, при необходимости, договориться об обсуждении с начальником или аналитиком, поставившим задачу. Цель данного взаимодействия — достижение общего понимания как задачи, так и метода её решения. Если начать реализацию без понимания всех деталей задачи, то велика вероятность, что решение окажется нерелевантным и его придётся переделывать целиком.

Иногда для определения метода решения задачи недостаточно только обсуждения и нужно провести дополнительную исследовательскую работу. Например, проанализировав документацию программной библиотеки, можно принять мотивированное решение о её использовании в проекте. Создание прототипа поможет удостовериться в правильности выбранного подхода.

Источником информации по новой задаче также может быть сообщение об ошибке от тестировщиков или конечных пользователей. При работе с таким материалом весьма распространённая ошибка — отрицание разработчиком самой возможности возникновения ошибки. Вместо этого необходимо собрать контекстную информацию: в чём состоит ошибка, при каких условиях или в каких сценариях использования она возникает, как можно её воспроизвести. Затем следует сформировать план по решению задачи, который, при необходимости, нужно обсудить со всеми вовлечёнными сторонами.

2.2. Типичные задачи проектирования, разработки и сопровождения исходного кода программного обеспечения

Разработка архитектуры (проектирование). После постановки задачи разработчик формирует архитектуру будущего программного кода. На базовом уровне она должна описывать способ организации компонентов, их входы и выходы, протоколы взаимодействия.

На архитектуру компонентов влияют следующие факторы:

— длительность жизненного цикла разрабатываемых компонентов;
— факт наличия в кодовой базе похожего решения;
— необходимость расширения или модификации компонентов в дальнейшем.

При разработке небольшой программы, которая будет запущена всего несколько раз, уровень детализации архитектуры, как правило, не имеет особого значения. Однако при создании нового компонента популярного веб-фреймворка архитектуру необходимо тщательно проработать и обсудить с другими разработчиками, чтобы получить универсальное и расширяемое решение.

Реализация программных модулей. На этапе написания программного кода необходимо не только обеспечить его корректность и эффективность, но и обеспечить удобство его восприятия. Это связано с тем, что на практике разработчики тратят много времени на чтение как своего, так и чужого исходного кода.

Для удобства поддержки изменений в исходном коде надо следовать принятым конвенциям по его оформлению: использовать

понятные имена переменных, классов и методов, по возможности добавлять и поддерживать в актуальном виде документационные комментарии. Понятный исходный код позволяет:

— уменьшать накладные расходы на взаимодействие членов команды;
— вносить изменения в код всей команде, а не только его автору.

Если исходный код не разделён на компоненты или плохо подготовлен для восприятия и внесения изменений, то скорость добавления новых функций будет низкой, а их стоимость, соответственно, высока.

Написание модульных тестов. Модульные тесты проверяют поведение отдельных модулей, тем самым обеспечивая защиту от ошибок. Когда разработчик добавляет новую функцию, автоматические тесты позволяют проверить отсутствие регрессии, то есть корректность работы уже реализованных функций.

Помимо этого, тесты являются хорошим средством документирования, так как заключают в себе основные сценарии использования компонента. Уже написанный код сценариев разработчики могут использовать в качестве основы при написания своего кода. Грамотное использование автоматического тестирования позволяет ускорить разработку и облегчить сопровождение.

Системное тестирование. По окончании всей разработки (или её итерации) необходимо удостовериться, что поставленные задачи были решены корректно с точки зрения пользователя. Во время тестирования стоит проверить как основной, так и альтернативные сценарии использования функций, подразумевающие ввод неверных значений, нарушение последовательности действий и так далее.

Может показаться, что системным тестированием должны заниматься исключительно тестировщики, тем самым сэкономив время разработчиков для решения других задач. Однако при такой организации работ скорее всего потребуется выполнить несколько циклов тестирования для успешного завершения реализации функции, что увеличит время её доставки.

Исправление ошибок. При получении такой задачи разработчику необходимо сначала воспроизвести указанную проблему. Если информации в постановке задачи недостаточно, тогда специалисту следует обратиться за разъяснениями.

После успешного исправления ошибки желательно проверить работу ключевых сценариев, зависящих от исправленной части приложения. Для документирования проблемной ситуации рекомендуется оформить автоматические тесты: они уберегут от повторения данной ситуации в будущем.

Написание документации. Для эффективной передачи знаний внутри команды разработчиков необходимо документировать ключевые решения проекта:

— из каких частей состоит система;
— как организованы потоки данных между компонентами и внешними системами;
— какие технологические и архитектурные решения были приняты и почему;
— каким образом устроены программные компоненты и как следует их использовать.

Эту информацию обычно оформляют в виде документов в выделенном хранилище знаний и в виде документационных комментариев к исходному коду.

Важно поддерживать массив знаний проекта в актуальном состоянии. Если разработчик столкнулся с некорректной информацией, то её по возможности необходимо исправить или сообщить о существующей проблеме.

Сопровождение проекта. После фазы активной разработки приложение переходит в фазу поддержки, при которой его ключевыми характеристиками становятся надёжность и бесперебойность работы. При решении задач в данной фазе разработчик должен не только реализовать требуемый функционал, но и приложить усилия по улучшению исходного кода, чтобы на следующую доработку этой части приложения потребовалось меньше времени. Правки могут быть косметическими (переработка названий переменных и методов, добавление документационных комментариев), структурными (разделение компонентов на части, переход на актуальные версии библиотек) и функциональными (добавление новых функций).

2.3. Обсуждение вопросов с коллегами

Зачастую разработчик прибегает к помощи коллег, когда не может самостоятельно продвигаться в решении задачи из-за недостат-

ка информации или отсутствия необходимого опыта. При неправильной организации обсуждения вопросов могут сформироваться неэффективные подходы к решению задач, когда каждый сотрудник работает изолированно, не обращается за помощью к коллегам и не стремится им помогать. Это приводит к увеличению сроков реализации функций, так как каждому разработчику необходимо самостоятельно разбираться во всём массиве информации по проекту.

Причинами неэффективного взаимодействия могут служить:

— неправильно выбранное время и место для вопроса;
— недостаточно конкретный вопрос, не содержащий деталей, которые необходимы для того, чтобы на него можно было ответить;
— злоупотребление вопросами при решении текущих задач.

Рассмотрим, каким образом можно сделать общение между коллегами продуктивнее.

Когда пора задавать вопрос? При возникновении проблемы разработчику следует избегать двух крайностей: немедленно обращаться за помощью или самостоятельно разбираться несколько дней. Можно потратить разумное время (полчаса или час) на поиск в сети Интернет, в исходном коде и документации. Если в результате приложенных усилий не удалось продвинуться в решении, пора формулировать вопрос к коллегам.

Как сформулировать вопрос? Разработчику необходимо детально описать проблемную ситуацию, а именно:

— условия возникновения проблемы;
— желаемый результат и фактическое поведение;
— опробованные подходы к решению.

Может показаться, что в этом случае разработчик потратит слишком много времени на подготовку, однако при обсуждении с коллегой эти вопросы неизбежно возникнут. Более точное описание проблемы ускорит её понимание и окупит приложенные усилия за счёт более быстрого нахождения решения.

Кому задавать вопрос? Проблема поиска коллеги, способного помочь с вопросом, в основном актуальна для новых сотрудников, так как они не знают сферы ответственности и компетенции остальных членов команды. Рассмотрим типичные способы решения данной задачи.

Во-первых, сотрудник может обратиться за помощью к руководителю группы, который в курсе специализации работников и их текущей нагрузки. Во-вторых, можно кратко обсудить вопрос на координационных встречах команды и договориться о дальнейшем личном обсуждении со специалистом. В-третьих, каждому сотруднику может быть предоставлен личный наставник, в ответственность которого входит передача информации об устройстве компании и о способах выполнения типичных задач, ответы на базовые вопросы.

Каким образом задавать вопрос? Наиболее эффективным способом взаимодействия является общение лицом к лицу, при котором можно вести живой диалог и совместно рассматривать проблемные участки исходного кода и другие материалы. Важно понимать, что при таком обсуждении на задачу тратится время сразу нескольких людей и его следует использовать максимально эффективно: к встрече надо подготовиться и заранее договориться о времени её проведения, чтобы всем участникам было удобно включить её в рабочий график.

Другой вариант обсуждения — это использование электронной переписки. В таком формате необходимо подготовить вопрос, структурированно подать контекстную информацию, добавить выдержки проблемного исходного кода или прямые ссылки на него. Коллеги смогут обработать запрос в удобном для себя режиме и дать комментарии. Не стоит рассчитывать на моментальный ответ, даже если общение происходит с помощью системы обмена мгновенными сообщениями.

Как отвечать на вопрос? Если коллега обращается с плохо поставленным вопросом, то по возможности следует обсудить подходы к его формулировке, чтобы сделать последующее взаимодействие продуктивнее.

На хорошо поставленный вопрос желательно дать развёрнутый ответ и обсудить проблему в широком контексте, чтобы выяснить её причину и снизить вероятность возникновения подобных ситуаций в будущем. Появление проблемы может указывать на системные недочёты в приложении, на которые следует обратить внимание всей команде.

2.4. Методы и техники организации работы разработчика

Для организации работы внутри команды зачастую используют трекер — специализированное ПО, в котором ведётся список задач. Обычно у них есть название, описание, исполнитель и статус. Название и описание содержат информацию о желаемом результате выполнения задачи. Статус указывает, на каком этапе решения находится задача: поставлена, принята в работу, приостановлена до получения дополнительной информации, передана в отдел тестирования, выполнена.

Корректное использование трекера позволяет снизить нагрузку на разработчиков, поскольку из него можно узнать, какими задачами сейчас заняты исполнители, на каком этапе находится конкретная задача, сколько задач сделано командой из списка запланированных и так далее.

Обычно количество задач, назначенных конкретному разработчику в трекере, значительно. Кроме того, в течение дня ему приходится переключать внимание на общение с коллегами, заниматься административными вопросами, отвечать на телефонные звонки и электронную почту, участвовать в совещаниях и так далее. Чтобы не утонуть в повседневной деятельности, важно рационально распределять время на разные её виды. Далее мы рассмотрим ряд методов для управления задачами, которые могут быть полезны разработчикам для повышения их личной эффективности.

Метод матрицы Эйзенхауэра. Все задачи должны быть распределены по критериям важно/не важно и срочно/не срочно на четыре категории. Примеры распределения задач разработки по категориям приведены в таблице 2.4. Выполнять задачи следует слева направо и сверху вниз.

Метод доведения дел до завершения (Getting Things Done, GTD) был разработан Дэвидом Алленом и описан в соответствующей книге [4]. Он включает в себя подход не только к распределению времени, но и к обработке информации, поступающей в течение всего дня. Метод GTD выделяет следующие активности: сбор информации, обработка входящих задач, организация списков, выполнение задач и анализ выполненных задач.

Таблица

Матрица Эйзенхауэра с примерами задач

	Срочно	*Не срочно*
Важно	Исправление критической ошибки при обработке банковских транзакций. Решение проблемы в прошивке целевого оборудования проекта вместе с его производителем.	Разработка архитектуры и программного интерфейса плагинов. Разработка системы непрерывной интеграции и развёртывания веб-приложения.
Не важно	Исправление неправильного логотипа веб-сайта. Написание информационного сообщения о выпуске очередного обновления приложения.	Переработка пользовательского интерфейса бизнес-приложения на основе новых технологий.

Для сбора информации о новых задачах используется концепция почтового ящика. Когда приходит новая задача или поступает новая информация, человек не отвлекается на её обработку, а лишь помещает в ящик «входящие» (inbox).

На этапе обработки содержимое «входящих» делится на собственно задачи и информацию, которая отправляется в архив или базу знаний. Быстрыми задачами, требующими не более нескольких минут, надо заняться немедленно, а остальные либо делегировать, либо запланировать их выполнение. Регулярно должна проводиться реорганизация списков с учётом срочности и актуальности задач.

В рабочее время необходимо выполнять задачи из списков в соответствии с выстроенным порядком. При выполнении важно учитывать, решение каких задач в настоящий момент будет наиболее эффективным. Для этого используются следующие критерии: местонахождение исполнителя, количество сил, времени и наличие необходимых инструментов.

Метод помидора [13] полезен для выполнения задач, требующих максимальной концентрации. Рабочее время разбивается на интервалы по 25 минут с перерывами по 5 минут между ними. Че-

рез каждые четыре интервала выполняется более долгий перерыв, до 30 минут, для восстановления сил.

Перед началом работы по методу помидора необходимо составить список задач для решения. Желательно, чтобы за один интервал запланированная задача решалась полностью: большие разбиваются на подзадачи, а небольшие объединяются.

Во время интервала нельзя отвлекаться, все оповещения от почты и мгновенных сообщений следует обработать во время перерыва или специального интервала. Если во время решения задачи приходит понимание, что необходимо выполнить новую задачу, то её следует записать в список и обработать согласно общему правилу.

Принцип девяти дел (принцип «1–3–5») предлагает за день выполнить одно большое дело, три средних и пять небольших. В этот набор входят как рабочие, так и важные домашние дела. Выделение большого дела позволяет сфокусировать усилия на его решении.

План на день составляется заранее и по возможности исполняется. На этапе планирования происходит выделение действительно важных задач, ненужные и менее важные дела не включаются. В конце дня легко подвести итог выполненным делам, проанализировать эффективность распределения времени.

Техника Fresh or Fried. В основе этой техники лежит предположение о том, что человек в разные части своего дня продуктивен по-разному. В зависимости от продуктивности задачи и распределяются: наиболее важные задачи ставятся на время наибольшей продуктивности, а на другое время планируются задачи менее приоритетные. Конкретное время наибольшей производительности может быть связано как с биологическими особенностями, например разделением на сов и жаворонков, так и с особенностями работы, например, в компании совещания могут всегда планироваться на первую половину дня.

2.5. Вопросы для обсуждения на семинарах

1. Может ли разработчик сразу перейти к реализации задачи без её обсуждения с менеджером? Обоснуйте свой ответ, приведите примеры.

2. С чем может быть связано неправильное понимание разработчиками функциональных спецификаций, написанных аналитиками?
3. Следует ли реагировать на малоинформативные сообщения об ошибках, в которых пользователь не даёт информации о контексте возникновения ошибки?
4. Чем отличается деятельность разработчиков, занимающихся реализацией проектов различных типов: прототипа продукта, типового продукта, уникального продукта с длительным циклом поддержки?
5. Когда стоит писать модульные тесты: до написания исходного кода, после или вообще не писать? Как это влияет на сам процесс разработки?
6. Когда в ходе разработки стоит выполнять оптимизацию исходного кода, вкладываться в поиск и реализацию самого эффективного алгоритма?
7. Когда в ходе разработки необходимо разбираться в своём или чужом исходном коде? Как вы думаете, насколько часто возникают такие ситуации?
8. Каким образом должен вести себя опытный член команды, когда к нему обращается начинающий разработчик?
9. Следует ли разработчику тратить значительную часть своего времени на улучшение качества исходного кода, а не на разработку новых функций приложения?
10. Может ли система управления задачами полностью заменить общение между членами команды?
11. Следует ли использовать только электронные средства общения, чтобы минимизировать отвлекающие факторы во время работы?
12. Использовали ли вы какие-либо методы управления рабочим временем и какой был результат их применения?
13. Какие методы управления рабочим временем могут быть использованы совместно?

Для подготовки к семинарам рекомендуется использовать следующую литературу: [2; 6; 9; 18; 19].

3. Деятельность менеджера в ИТ-компании

3.1. Менеджер и его функции

Менеджер — это должность или роль сотрудника, основные функции которого лежат в сфере управления. Под управлением понимается деятельность, направленная на создание и поддержание на предприятии внутренней среды, позволяющей группам сотрудников совместно и эффективно двигаться к достижению поставленных целей. Основные функции управления следующие:

— планирование (постановка целей, анализ рисков, оценка ресурсов);
— организация (определение полномочий и ответственности);
— кадровое обеспечение;
— управление подчинёнными;
— регулирование (оценка и корректировка деятельности).

В области ИТ можно условно выделить менеджеров двух типов: «технических» и «нетехнических». Данные термины не являются общепринятыми, однако достаточно точно выражают мысль авторов: «технический» менеджер, в отличие от «нетехнического», преимущественно управляет техническими специалистами (разработчиками, тестировщиками и т. д.) и принимает решения, непосредственно связанные с информационными технологиями и организацией процесса разработки. Рассмотрим основные должности (роли) менеджеров этих типов.

Руководитель группы (team lead) — это менеджер, управляющий группой разработчиков. Его основные функции:

— обработка (иногда и получение) требований от заказчика;
— организация процесса разработки в команде;
— оперативное управление разработчиками, тестировщиками, дизайнерами;
— рецензирование программного кода;
— управление версиями продукта;
— промежуточная приёмка продукта.

Менеджер проекта (project manager) — это менеджер, управляющий конкретным проектом или проектами в компании. Его основные функции:

— разработка плана управления проектом;

— определение содержания проекта;
— планирование проекта и управление рисками;
— оценка и управление ресурсами проекта;
— управление работами проекта и их мониторинг;
— контроль бюджета и сроков проекта.

Менеджер по продукту (product manager) — это менеджер, осуществляющий высокоуровневое управление полным циклом разработки конкретного продукта или семейства продуктов. Его основные функции:
— анализ рынка и формирование идеи продукта;
— управление созданием прототипа продукта и его оценкой;
— осуществление коммуникации между всеми задействованными в ходе разработки продукта специалистами;
— переговоры с заказчиками;
— продвижение продукта проекта на рынке.

Технический директор — это менеджер, ответственный за организацию всех технических специалистов и высокоуровневое управление процессом разработки программного обеспечения в масштабе всей компании. Его функции:
— руководство всеми этапами создания и сопровождения продукта на протяжении всего его жизненного цикла;
— координация работы всех ИТ-специалистов, распределение обязанностей и ответственности между ними;
— разработка и реализация стратегии технического и технологического развития компании;
— организация обучения ИТ-специалистов.

Исполнительный директор, генеральный директор — высшие должности в руководстве компании. Их основные функции:
— разработка и внедрение стратегических планов компании;
— организация экономической деятельности;
— организация взаимодействия всех подразделений компании;
— мониторинг и коррекция всех видов деятельности в компании;
— контроль за соответствием деятельности компании действующему законодательству.

3.2. Типы менеджеров в модели PAEI

Всё многообразие функций, выполняемых менеджерами, на практике может осуществляться различными способами. Адизес [3] в своей модели PAEI выделяет четыре роли менеджера, характеризующие различные стили управления:

— P — производитель (producer);
— A — администратор (administrator);
— E — предприниматель (entrepreneur);
— I — интегратор (integrator).

Рассмотрим эти роли подробнее.

Производитель — это менеджер, ориентированный на получение результатов в краткосрочной перспективе. Он старается обеспечить решение всех текущих задач, причём зачастую своими собственными силами. В сфере ИТ такой менеджер в прошлом является квалифицированным техническим специалистом и, в силу этого, тяготеет к самостоятельному решению задач разработки, а не к делегированию их своим подчинённым. Слабостью такого менеджера может быть недостаточное стратегическое мышление.

Администратор — это менеджер, нацеленный на систематизацию процессов управления и оптимизацию использования всех имеющихся ресурсов. Такой менеджер всегда наводит порядок в своей зоне ответственности, однако склонен к бюрократизации процесса и зачастую избегает риска, так как тот способен внести дезорганизующий элемент в его стандартизованные процедуры.

Предприниматель — это менеджер, устремлённый в будущее, который всегда старается генерировать новые идеи и немедленно их внедрять. Он стремится к постоянному развитию и толкает своих сотрудников к саморазвитию, «заражая» их своим энтузиазмом. Слабостями такого типа менеджера могут быть склонность необоснованно рисковать и не доводить до конца те идеи, к которым он уже «остыл».

Интегратор — это менеджер, ориентированный на организацию процессов в долгосрочной перспективе за счёт создания общих традиций, ценностей, корпоративной культуры. Он сплачивает коллектив вокруг себя, стараясь обеспечить сотрудникам такие условия работы, которые позволили бы им работать вместе эффективно. Такой менеджер обычно очень вежлив и предупредителен со своими сотрудниками, однако может быть с ними излишне мягким.

Ни один менеджер не может сочетать в себе все четыре роли в полной мере. На практике у успешного менеджера одна из ролей всегда доминирует, а все остальные он выполняет, как минимум, на некотором элементарном уровне. Таким образом, складывается некоторый баланс, являющийся основой индивидуального управленческого стиля («профиля менеджера»). Невозможность выполнять хотя бы одну из перечисленных ролей существенно снижает шансы руководителя на успех, поскольку у любого менеджера обязательно есть функции, выполнение которых требует каждой из перечисленных ролей.

Понимание профиля полезно для менеджера, так как позволяет ему знать свои сильные и слабые стороны и гармонично развиваться в профессиональном плане. Также это понимание полезно и для его сотрудников, поскольку позволяет лучше понимать менеджера и успешнее взаимодействовать с ним.

3.3. Взаимопонимание менеджера и разработчика

В ходе процесса разработки менеджеры и разработчики постоянно взаимодействуют друг с другом, и от эффективности этого взаимодействия зачастую зависит успех конкретного проекта или всего процесса разработки. К сожалению, в силу некоторых причин, это взаимодействие зачастую представляет трудности для обеих сторон и сопровождается взаимными упрёками или даже конфликтами.

Основное противоречие, являющееся причиной непонимания между разработчиком и менеджером, заключается в том, что их системы ценностей и точки зрения на процесс разработки различны. Для разработчика целевыми профессиональными установками могут быть оригинальные архитектурные решения, применение новых технологий или библиотек, красиво и качественно написанный код. Для менеджера, в свою очередь, интерес представляют соблюдение календарного плана, удовлетворение потребностей заказчика, быстрое исправление ошибок в продукте.

В результате достаточно часто встречается ситуация, когда разработчик, испытывающий гордость за результат своей работы, объясняет менеджеру, как изящно он решил трудную техническую проблему, ожидая поощрения, а менеджер вместо этого указывает ему

на чрезмерное количество времени, которое заняла реализация этого решения, а также на наличие большого количества ошибок в продукте, требующих срочного исправления. И это только один пример, демонстрирующий возможные трения между разработчиком и менеджером.

Решение указанной проблемы на практике обычно является весьма непростым, потому что оно требует от обеих сторон систематически рассматривать свою текущую работу с точки зрения противоположной стороны. Это означает, что разработчик не должен чрезмерно увлекаться технической стороной процесса разработки, «игрой» с технологиями и быть перфекционистом по отношению к своему и чужому коду. Ему необходимо научиться видеть потребности заказчика за функциональными требованиями и хотя бы в некоторой степени разбираться в предметной области каждого проекта. Также немаловажно для разработчика следить за календарными планами и информировать менеджера в случаях, когда их соблюдение оказывается под вопросом.

В свою очередь, менеджер должен понимать, что разработчик обладает важной информацией о ходе проекта, которая может быть незаметной снаружи. Кроме того, не всегда перфекционизм разработчика является неоправданным, иногда он бывает обусловлен желанием решить внутренние проблемы проекта, которые разработчик видит, но, возможно, не может точно выразить. Хороший менеджер должен уметь выделять такие ситуации, так как именно в его распоряжении находятся инструменты, позволяющие более эффективно их разрешать на организационном уровне, тогда как менеджер, который не прислушивается к разработчикам, лишает себя важного инструмента управления рисками.

3.4. Лидерство менеджера

Лидерство — способность человека оказывать влияние на отдельных людей и группы, направляя их усилия на достижение некоторых целей. Лидерство базируется на личностных качествах человека, на его авторитете и проявляется в первую очередь в умении вести других людей за собой.

Несмотря на определённое сходство между управлением и лидерством (оба связаны с обладанием властью и нацелены на её приме-

нение для того, чтобы заставить других людей достигать определённых целей), между ними есть существенное отличие: руководителю члены команды обязаны подчиняться, тогда как лидер ведёт за собой других людей, а те выступают по отношению к нему не столько как подчиненные, сколько как последователи.

Таким образом, менеджер совсем не обязательно должен быть лидером, и наоборот: сотрудник, не являющийся менеджером, может играть в коллективе роль неформального лидера. В частности, таковыми являются профсоюзные и общественные деятели. Иногда даже случается так, что между руководителем и неформальным лидером в коллективе возникает борьба за власть, которая, безусловно, отрицательно сказывается на эффективности работы коллектива в целом. Но возможен случай, когда менеджер в коллективе сам является лидером. Это происходит, когда менеджер обладает необходимыми личными качествами и последовательно придерживается некоторых общих принципов лидерства для руководителя. Рассмотрим их подробно.

— Лидер должен чётко видеть перспективу и понимать, чего он хочет достичь и куда привести свою команду. Без этого он не сможет быть достаточно убедительным со своими последователями и не сможет вести их за собой.
— Руководитель-лидер не должен, кроме исключительных ситуаций, использовать формальную власть руководителя. Вместо этого он должен опираться на свои лидерские качества и реализовывать способность личного влияния на других людей. За счёт этого достигается более мягкая форма управления, в рамках которой сотрудники работают без явного принуждения, на доверии к своему лидеру. Применение формальной власти может разрушить это доверие.
— Руководитель-лидер должен делиться знаниями со своими сотрудниками и быть для них наставником в профессиональной сфере, а также культивировать наставничество среди своих наиболее квалифицированных специалистов. Данный принцип обеспечивает, с одной стороны, атмосферу взаимопомощи в команде, а с другой — постоянное развитие команды и профессиональное самосовершенствование её членов.
— Руководитель-лидер должен защищать свою команду от неблагоприятных внешних факторов. В число последних могут вхо-

дить, например, недовольство вышестоящего начальства, непонимание со стороны заказчиков, неблагоприятная конъюнктура рынка. И, напротив, совершенно недопустимо для лидера перекладывать ответственность за неудачи команды на своих сотрудников.

— Руководитель-лидер должен быть примером для своих сотрудников. Это важно, потому что между лидером и его последователями устанавливается тесная личностная связь, в результате чего последователи начинают, часто неосознанно, копировать стиль поведения своего лидера. В результате возможные отрицательные качества лидера начинают проявляться и у его последователей. Ещё хуже, когда лидер сам не следует тем принципам, которые продвигает среди последователей, поскольку такой лидер очень легко потеряет свой авторитет.

3.5. Вопросы для обсуждения на семинарах

1. Оцените, какие риски будет нести проект и организация, если менеджер окажется не способен исполнять перечисленные в разделе 3.1 функции должным образом.
2. Соотнесите перечисленные в разделе 3.1 должности с классификацией менеджеров как «технических» и «нетехнических».
3. Кто находится в подчинении перечисленных в разделе 3.1 типов менеджеров? Есть ли среди них другие менеджеры?
4. Должен ли руководитель группы самостоятельно решать задачи по разработке ПО или только заниматься их распределением среди других специалистов?
5. Как соотносятся роли аналитика проекта и менеджера по продукту?
6. Для каждой из перечисленных в разделе 3.2 ролей ответьте на следующие вопросы, касающиеся менеджера, у которого эта роль является доминирующей:
   — Как к такому менеджеру относятся сотрудники?
   — Как правильно себя вести сотрудникам по отношению к такому менеджеру?
   — Как такой менеджер организует совещания?

— Насколько вероятно, что такой менеджер станет лидером в коллективе разработчиков? Благодаря каким качествам, свойственным соответствующим ролям, это может произойти?

7. Сформулируйте наилучший тип менеджера, согласно модели из раздела 3.2, для каждой из ролей менеджера, перечисленных в разделе 3.1.
8. Сформулируйте ваши ключевые профессиональные ценности как разработчика программного обеспечения. Может ли их последовательное продвижение стать причиной недопонимания с менеджером? Как его можно избежать?
9. Рассмотрите, какие подходы в распространённых методологиях разработки приложений (Scrum, Kanban, экстремальное программирование) направлены на решение конфликтных ситуаций в коллективе.
10. Сформулируйте преимущества, которые получают компания в целом и её сотрудники, если её руководитель является лидером.
11. Объясните, как именно руководитель-лидер может защищать свою команду от неблагоприятных факторов.
12. Конкретизируйте принципы лидерства для руководителя из раздела 3.4 применительно к сфере разработки программного обеспечения.

Для подготовки к семинарам рекомендуется использовать следующую литературу: [3; 5; 8; 14; 15].

4. Карьерный путь разработчика

4.1. Варианты самореализации в ИТ-индустрии

Можно выделить три основных варианта самореализации специалиста в ИТ-индустрии: в качестве сотрудника по найму, фрилансера и предпринимателя. Рассмотрим их более подробно.

Вариант работы в качестве сотрудника по найму предполагает трудоустройство в некоторой компании на определённую должность (разработчик, тестировщик и т. д.). Наёмный сотрудник, как

правило, имеет в компании заранее определённый круг обязанностей и зону ответственности, за пределы которых он не должен выходить, и постоянную, заранее оговорённую зарплату и оплачиваемый отпуск. Карьерный рост возможен, но только в рамках потребностей компании в тех или иных специалистах. Отметим, что последнее не влечёт за собой ограниченности карьерного роста для конкретного специалиста в принципе, а лишь может означать отсутствие возможностей для него в рамках одной конкретной компании. В совокупности все перечисленные факторы определяют то, что подавляющие большинство специалистов строит свою карьеру именно как наёмные работники.

Фрилансер (по английски слово freelancer означает «свободный художник») — это специалист, который не работает на конкретную компанию, а самостоятельно предлагает свои услуги различным заказчикам. Большинство фрилансеров работают удалённо. С одной стороны, такой вариант самореализации предоставляет бóльшую степень свободы сотруднику в вопросах выбора заказчиков, проектов, графика работы и направления собственной специализации. С другой стороны, работа фрилансером требует бóльших усилий от сотрудника и более высокой квалификации по сравнению с работой по найму. В первую очередь это связано с тем, что фрилансер должен предпринимать усилия для самостоятельного нахождения проектов, иметь навыки ведения бизнеса и более высокий уровень технической компетентности, чтобы быть конкурентоспособным на рынке фриланса.

Предпринимательство — это наиболее сложный и сопряжённый с наибольшими рисками путь построения карьеры, связанный с самостоятельным формированием инициативных проектов и организацией процесса их разработки. Его особенность состоит в том, что разработчик во время своего становления в качестве предпринимателя постепенно из технического специалиста превращается в специалиста по бизнесу, делегируя решение технических задач наёмным работникам. Главное преимущество предпринимательской карьеры — возможность полностью самостоятельного принятия всех решений, касающихся профессиональной деятельности. Оно же является недостатком: ответственность за эти решения также придётся нести самостоятельно, причём сюда входит также ответственность за своих наёмных сотрудников.

Следует отметить, что среди перечисленных вариантов самореализации нет «хороших» или «плохих», и выбор конкретного пути самореализации должен быть обусловлен в первую очередь личными качествами и стремлениями ИТ-специалиста, и каждый должен выбирать тот вариант, который был бы комфортным лично для него. Что же касается уровня заработной платы, то нужно понимать, что он зависит не от выбранного варианта самореализации, а от того, сколько сил было вложено конкретным специалистом в своё профессиональное развитие и насколько успешным оно оказалось. И вопреки распространённому заблуждению, в индустрии встречаются и наёмные специалисты с очень высокой заработной платой, и постоянно балансирующие на грани банкротства предприниматели.

4.2. Возможные пути развития специалиста в иерархии компании и по ступеням профессионального мастерства

В разделе 1.2 были рассмотрены те роли, которые могут играть сотрудники в компаниях по разработке программного обеспечения. В рамках этих ролей специалисты могут двигаться по ступеням профессионального мастерства, переходя от более простых, локальных задач к более сложным задачам, решение которых имеет существенное значение как для проектов, реализуемых ИТ-компанией, так и для всей компании в целом.

Как правило, в крупных компаниях движение по «лестнице» профессионального мастерства означает также карьерный рост, т. е. переход к более высоким должностям. Например, стажёр со временем может стать младшим разработчиком, затем ведущим разработчиком, затем руководителем отдела разработки. Иерархия должностей в каждой компании различна, поэтому задачи в этом разделе не сгруппированы по конкретным должностям. В компаниях малого и среднего бизнеса, особенно в тех, где используется гибкая методология разработки, один сотрудник может играть несколько ролей, поэтому там не обязательно наличие формальных должностей для разных уровней профессионального развития. Тем не менее общая идея соответствия степени профессионального мастерства и положения в компании сохраняется.

Перечислим теперь типовые профессиональные задачи, сгруппировав их по ролям и расположив в порядке увеличения сложности и значимости. При этом нужно иметь в виду, что некоторые роли сознательно сгруппированы попарно, так как переход от одной роли к другой характерен для некоторых типичных схем профессионального роста (например, превращение разработчика в архитектора). Кроме того, на высоких ступенях профессионального развития специалисты часто получают те или иные менеджерские функции, не являясь при этом выделенными менеджерами.

Роли «разработчик», «архитектор»:

— преобразование готовых спецификаций в программный код;
— сопровождение существующего программного кода;
— построение модульных спецификаций;
— написание автоматически выполняемых тестов;
— документирование программного кода;
— рефакторинг программного кода;
— разработка и внедрение архитектурных решений;
— решение задач интеграции программного кода;
— делегирование задач другим разработчикам и контроль результатов решения этих задач;
— разработка спецификаций системного уровня, в т. ч. по требованиям пользователей;
— аудит безопасности, производительности и других системных свойств разрабатываемого ПО;
— взаимодействие с командами других специалистов (тестировщиков, дизайнеров, системных аналитиков и т. д.);
— разработка архитектуры на системном уровне;
— организация работы команды разработчиков.

Роли «тестировщик», «инженер по обеспечению качества»:

— ручное тестирование продукта по готовым тест-кейсам;
— подготовка и сопровождение тест-кейсов;
— разработка автоматизированных тестов;
— создание тестовых фреймворков;
— создание инфраструктуры для выполнения тестов;
— анализ требований к программному обеспечению;
— контроль качества продукта;

— планирование и реализация мероприятий по управлению качеством.

Роли «дизайнер», «проектировщик пользовательских интерфейсов»:

— выполнение художественно-оформительских работ;
— разработка и сопровождение эскизов пользовательского интерфейса (user interface, UI);
— разработка сценариев взаимодействия приложения с пользователем (user experience, UX);
— аудит удобства использования.

Роль «технический писатель»:

— разработка и сопровождение технической документации;
— сбор информации, подлежащей документированию, со всех участников процесса разработки;
— разработка регламентов и стандартов технической документации;
— внедрение средств разработки технической документации;
— управление разработкой технической документации.

Роль «системный администратор»:

— администрирование серверов и рабочих станций компании;
— осуществление технической поддержки пользователей;
— обеспечение бесперебойного функционирования инфраструктуры;
— организация закупок оборудования;
— планирование и масштабирование инфраструктуры;
— анализ рисков и обеспечение информационной безопасности.

Роль «аналитик» («системный аналитик», «бизнес-аналитик»):

— сбор и формализация требований заказчика;
— анализ потребностей рынка;
— разработка технических заданий и технико-экономических обоснований проектов;
— разработка планов внедрения продукта;
— прогноз и оценка эффективности маркетинговой деятельности;
— анализ рисков;
— разработка и реинжиниринг бизнес-проектов;
— информационная поддержка менеджеров.

Роль «менеджер» была подробно рассмотрена в главе 3.

Помимо развития в рамках одной специальности, возможен также переход специалиста с одной специальности на другую при формировании у него навыков, необходимых для такой переквалификации. Самый типичный пример — превращение тестировщика в разработчика в результате более глубокого усвоения им инструментальных средств и методов разработки программного обеспечения. Другой более редкий, но очень важный сценарий — переход разработчика к роли аналитика. Аналитик, прошедший подобный путь профессионального развития, обычно представляет особую ценность для компании, потому что хорошо понимает не только бизнес-цели компании и заказчиков, но и процесс разработки, что позволяет ему предлагать эффективные решения.

4.3. Интересы компании и сотрудника, их взаимная ценность друг для друга

Одним из главных факторов, определяющих, насколько комфортной и плодотворной будет работа сотрудника в компании, является фактор взаимной ценности сотрудника и компании. Главная идея здесь состоит в том, что сотрудник и компания должны быть примерно в одинаковой степени заинтересованы друг в друге. Несоблюдение этого правила приводит к тому, что одна из сторон начинает испытывать неудобства от рабочего процесса, что со временем может повлечь прекращение отношений между сотрудником и компанией по инициативе той или иной стороны. Рассмотрим типичные проявления несоответствия между ценностью компании и сотрудника друг для друга.

Пример, когда сотрудник в большей степени заинтересован в работе в компании, чем компания в нём, — студент второго-третьего курса, который устроился на своё первое место работы. Он высоко мотивирован работать в компании и держится за своё место, в то время как для компании его ценность весьма невелика: у него нет опыта работы, его нужно многому учить, продуктивность и качество его работы зачастую низкие. Из-за очевидного дисбаланса положение дел таким долго оставаться не может, и либо студент прогрессирует в профессиональном плане, повышая свою ценность для компании, либо компания избавляется от такого сотрудника по

какой-либо формальной причине, например по истечении срочного трудового договора.

Пример обратной ситуации — в компании работает опытный сотрудник, который имеет более высокую квалификацию, чем это необходимо для задач, соответствующих его должности. Компания нуждается в его работе, однако в силу каких-то причин не предоставляет возможностей для карьерного роста и более успешного применения его квалификации. В силу этого сотрудник начинает испытывать ощущение, что его не ценят, загружают рутинной работой, не продвигают по службе, выплачивают ему слишком низкую заработную плату и т. д. В результате либо такой сотрудник уходит из компании в поисках лучшего места работы, либо руководство всё-таки осознаёт, что необходимо идти навстречу профессиональным пожеланиям такого сотрудника, если его ценность действительно велика для компании.

Для того чтобы повысить свою ценность, сотрудник должен чётко понимать и транслировать компании свои профессиональные установки. Их могут выражать, например, ответы на такие вопросы: «В каком направлении я хочу развиваться? Каковы мои цели? В какой сфере я хотел бы реализовать себя? Что меня вдохновляет работать с полной отдачей? Чему я хочу научиться?»

В свою очередь, компания должна учитывать интересы ценных для неё сотрудников при планировании их профессионального роста, разумеется, соотнося их с интересами самой компании. Основным подходом здесь является методика индивидуальных планов сотрудников.

Индивидуальный план сотрудника обычно включает в себя цели развития и мероприятия по развитию. Цели развития определяют, каких именно результатов предполагается достичь в течение периода, на который составлен план, например «достичь позиции инженера-тестировщика среднего звена». Типичная длительность периода обычно составляет полгода или год.

Мероприятия по развитию — это конкретные задачи, которые должен научиться выполнять сотрудник в рамках достижения целей своего развития. Например, «овладеть техниками проведения рефакторинга и применять их в текущих проектах». Для каждого мероприятия указываются сроки проведения, ответственные лица (чаще всего это сам сотрудник и его непосредственный начальник),

требуемые ресурсы (например, сотруднику может быть выделено 10 % рабочего времени на выполнение мероприятий по саморазвитию), а также показатели результативности. Последние по возможности должны быть измеримыми, чтобы по истечении периода можно было провести некоторое контрольное мероприятие (например, тестирование) или вычислить некоторую метрику, которая непосредственно характеризует текущую работу сотрудника (например, количество дефектов в коде сотрудника, которые были обнаружены не им самим, а отделом обеспечения качества), и сделать вывод о том, насколько успешно было выполнено конкретное мероприятие.

По окончании периода сотрудник во время встречи со своим непосредственным руководителем получает обратную связь об эффективности своей работы за период и выполнении целей по развитию. По результатам сотрудник может быть премирован или переведён на более высокую должность.

4.4. Процесс превращения разработчика в менеджера

Существуют различные причины, по которым компания может инициировать процесс перевода разработчика в менеджеры. Перечислим основные из них.

— Стремление «вырастить» менеджера внутри компании. Такой менеджер обычно имеет преимущество над взятым со стороны, т. к. глубже понимает специфику своей организации и отрасли, а также знает сильные и слабые стороны коллектива.
— Отсутствие возможностей для сотрудника развиваться внутри компании в качестве разработчика при наличии у него потенциала для развития.
— Отсутствие менеджера при невозможности увеличения штата сотрудников или нежелание брать человека со стороны по экономическим (недостаток финансирования или времени на «врабатывание» внешнего человека в процесс), а иногда и по политическим причинам.

Сам процесс превращения разработчика в менеджера обычно начинается с того, что разработчик получает под своё управление группу других разработчиков либо отдельный проект компании.

Собственно задачи, с которыми сталкивается менеджер, обсуждались в разделе 3.1, а в данном разделе сосредоточимся на тех отличиях в характере работы менеджера и разработчика, которые начинающий менеджер должен усвоить для того, чтобы эффективно и безболезненно освоить новую роль.

— В работе менеджера значительную роль и существенную долю рабочего времени занимает взаимодействие со всеми участниками процесса разработки, включая заказчика, а также руководство компании.
— В отличие от разработчика, которому необходимо максимально концентрироваться на своей работе, работа менеджера требует постоянного переключения между различными видами деятельности и принятия множества решений в течение рабочего дня.
— Если разработчик, как правило, несёт ответственность только за результат своей собственной работы, менеджер несёт ответственность за весь проект или всю команду в целом.
— В функциях менеджера выражен психологический аспект: менеджер мотивирует команду и обеспечивает её поддержкой.

В заключение рассмотрим ошибки, которые часто совершает начинающий менеджер, «переквалифицированный» из разработчика.

— Начинающий менеджер часто (и иногда вынужденно) совмещает функции разработчика и менеджера. Однако, в силу принципиально различного характера работы, основанного на быстром переключении между задачами, с одной стороны, и высокой концентрацией — с другой, результат такого совмещения часто оказывается неуспешным.
— Менеджер пытается выполнить всю работу самостоятельно. Обычно это бывает вызвано неумением обеспечить решение всех задач силами команды. Но, поскольку объём задач в этом случае существенно превышает возможности одного человека, такой подход заранее обречён на неудачу.
— Менеджер пытается уйти от ответственности за работу команды. Как правило, это вызвано непониманием своей новой роли в процессе разработки и в достижении целей компании.
— Чрезмерный контроль над работой подчинённых и постановка нереальных сроков для выполнения задач. Обычно обусловлены отсутствием контакта между менеджером и разработчиками, а также неправильной оценкой их сил и возможностей.

4.5. Вопросы для обсуждения на семинарах

1. Оцените, насколько распространён фриланс при разработке приложений различных типов (например, корпоративных информационных систем, мобильных приложений, веб-приложений, систем реального времени и т. д.). С чем это связано?
2. Какие нетехнические знания и навыки необходимо приобрести ИТ-специалисту, выбравшему вариант самореализации, связанный с предпринимательством?
3. Для парных ролей из раздела 4.2 (например, *разработчик — архитектор*) разделите перечисленные задачи на типичные для отдельных ролей и совмещаемые обеими ролями. Объясните, в каких случаях происходит совмещение. Определите, какие задачи должен решать специалист, чтобы перейти от первой роли пары ко второй.
4. Как может развиваться сотрудник, совершая переход к принципиально другой специальности (например, от дизайнера к разработчику)? При каких условиях такое развитие может быть возможно? Приведите примеры.
5. Что может происходить в ситуации, когда компания заинтересована в продвижении своего сотрудника, но сам сотрудник не обозначает для руководства компании направлений для предпочтительного развития? Какие последствия это может иметь для сотрудника?
6. Перечислите менеджерские задачи, которые появляются у разработчика, в случае когда он получает под своё управление группу других разработчиков. Раскройте содержание этих задач и опишите способы их решения.
7. В чём конкретно могут проявляться негативные последствия ситуации, в которой один человек совмещает функции разработчика и менеджера?

Для подготовки к семинарам рекомендуется использовать следующую литературу: [1; 10; 12; 16; 18].

Список литературы

1. *Gamell J.* From Engineer to Manager: keeping your technical skills. URL: *https://hackernoon.com/from-engineer-to-manager-keeping-your-technical-skills-40579cc8ea00* (visited on 07/26/2020).
2. *Kim M.* Programmer Vs Developer Vs Engineer: The Naming Dispute. URL: *https://medium.com/shakuro/programmer-vs-developer-vs-engineer-91ef374e5033* (visited on 07/26/2020).
3. *Адизес И.* Как преодолеть кризисы менеджмента. М. : Манн, Иванов и Фербер, 2014. 320 с.
4. *Аллен Д.* Как привести дела в порядок: искусство продуктивности без стресса. М. : Вильямс, 2007. 368 с.
5. *Архипенков С.* Руководство командой разработчиков программного обеспечения. Прикладные мысли. 2019. URL: *http : / / citforum.ru/SE/project/app_thoughts/* (дата обр. 26.07.2020).
6. *Буна С.* Не путайте разработку ПО и программирование. URL: *https:// habr.com / ru / company / alconost / blog / 341304/* (дата обр. 26.07.2020).
7. *Вест Д.* Роли Scrum и правда о должностях в Scrum. URL: *https://www.atlassian.com/ru/agile/scrum/roles* (дата обр. 26.07.2020).
8. Война за таланты. Как удержать ключевых сотрудников / Smart Business Solutions — практика современного подхода. URL: *https:// www.sbsc.ru / books / TalentWarBook.pdf* (дата обр. 26.07.2020).
9. *Гапешин Д.* 15 популярных техник тайм-менеджмента. URL: *https:// skillbox.ru / media / growth / 15_populyarnykh_tekhnik_taym_menedzhmenta/* (дата обр. 26.07.2020).
10. *Кашкин М.* Разработчик — проектный менеджер? URL: *https:// dou.ua / lenta / articles / programmer-vs-manager/* (дата обр. 26.07.2020).
11. *Львова А.* Какими бывают IT-компании // Клевер: журнал о работе в IT. URL: *https://klever.blog/types-of-it-companies/* (дата обр. 26.07.2020).

12. *Мироненко А.* Как строить карьеру: условия и варианты роста в компании // Клевер: журнал о работе в IT. URL: *https://klever.blog/how-to-build-a-career-part2/* (дата обр. 26.07.2020).

13. *Нётеберг Ш.* Тайм-менеджмент по помидору. Как концентрироваться на одном деле хотя бы 25 минут. М. : Альпина Паблишер, 2019. 246 с.

14. Нужны ли менеджеры в IT? URL: *https://habr.com/ru/post/219741/* (дата обр. 26.07.2020).

15. *Рейнвотер Д. Х.* Как пасти котов. Наставление для программистов, руководящих другими программистами. СПб : Питер, 2019. 256 с.

16. Свой бизнес в IT // Хакер. URL: *https://xakep.ru/2014/09/09/bizness-it/* (дата обр. 26.07.2020).

17. *Стельмах С.* Цифровая трансформация и ИТ-аутсорсинг: стратегия сотрудничества. URL: *https://www.itweek.ru/digitalization/article/detail.php?ID=211790* (дата обр. 26.07.2020).

18. *Фаулер Ч.* Программист-фанатик. М. : Питер, 2018. 208 с.

19. *Хант Э., Томас Д.* Программист-прагматик. Путь от подмастерья к мастеру. М. : Лори, 2009. 270 с.