Фергюс Хендерсон программирует без малого четыре десятилетия. Последние 10 лет он работает в Google, где теперь возглавляет команду разработчиков технологии перевода текста в речь. 31 января 2017 года Хендерсон опубликовал документ, «подробно описывающий лучшие практики разработки программного обеспечения в Google». dev.by подготовил полный перевод текста на русский язык.

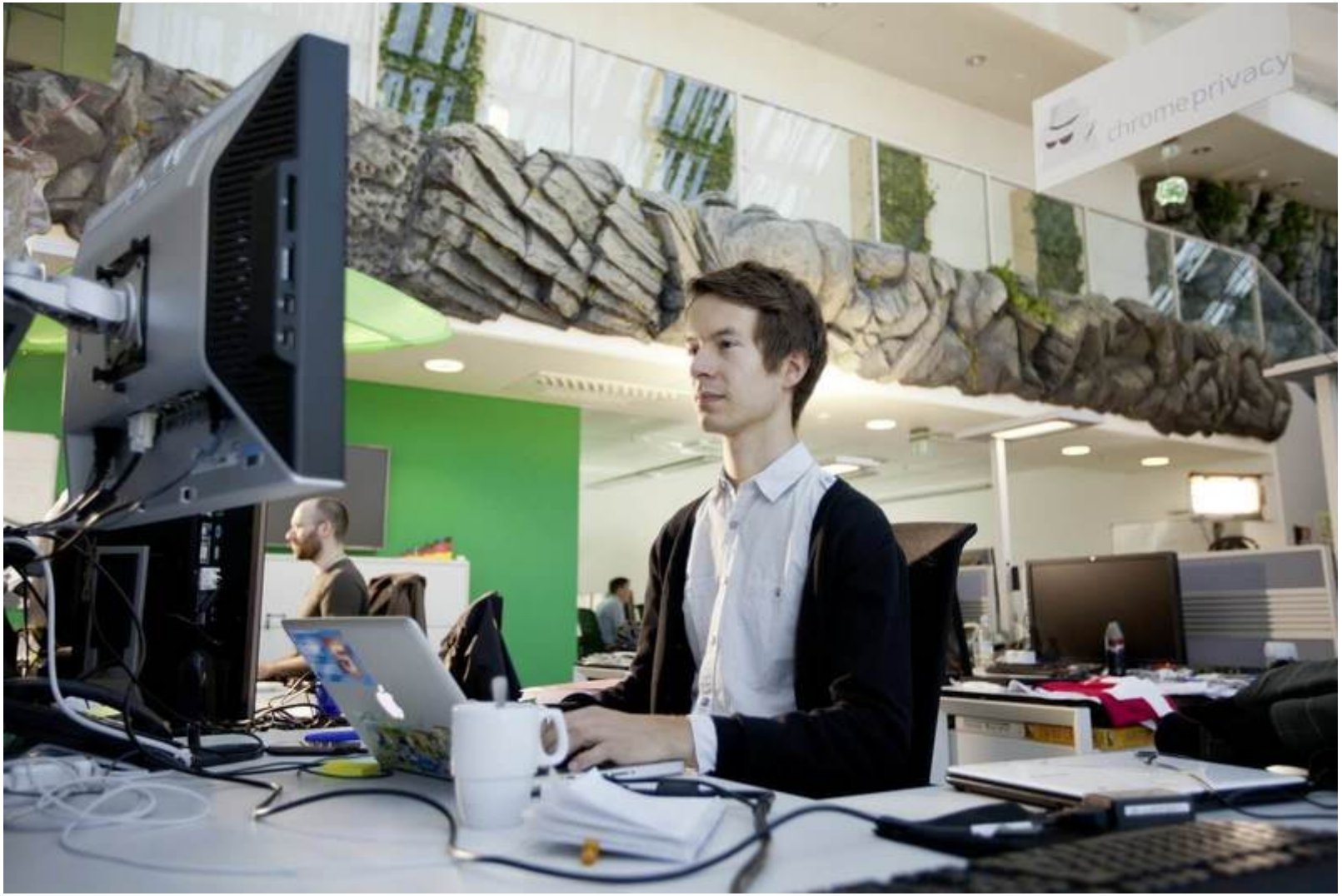


Фото: Google

1. Введение

Google всегда была крайне успешной компанией. Помимо поисковой системы и сервиса интернет-рекламы AdWords, можно смело перечислить ряд других ярких продуктов: Google Maps, Google News, Google Translate, сервис распознавания речи, браузер Chrome и операционная система (ОС) для мобильных устройств Android. Google также в значительной мере улучшила и масштабировала продукты мелких приобретённых компаний (например, YouTube), а также дополнила открытый исходный код ряда приложений. Но компания не останавливается на достигнутом и в ближайшей перспективе — запуск беспилотного автомобиля Google.

Успех Google можно объяснить рядом причин, среди которых компетентное руководство, талантливые люди, высокие требования при наборе сотрудников, а также финансовая стабильность — результат раннего лидерства на таком стремительно развивающемся рынке.

Однако основным ключом к успеху всё же являются отлаженные методы и подходы к разработке программного обеспечения. Они вырабатывались в течение долгого времени и основаны на лучшем опыте, накопленном самыми талантливыми разработчиками со всего мира.

Мы хотели бы поделиться этим опытом с миром, а также теми уроками, которые извлекли, совершая многочисленные ошибки на пути его накопления.

> Главная цель этой публикации — сохранить и описать ключевые методы разработки ПО в Google. Остальные компании и разработчики получат возможность сравнить их с методами, которые они привыкли использовать, и, возможно, внедрить что-то из того, что используем мы.

Многие авторы (например, [9], [10], [11]) уже анализировали историю успеха Google. Большинство из них изучали методы ведения бизнеса, менеджмента и корпоративную культуру; и только некоторые ([1], [2], [3], [4], [5], [6], [7], [13], [14], [16], [21]) пытались вникнуть в процессы организации разработки — и если делали это, то исследовали лишь очень узкий его аспект. Никто до этого не публиковал обзор основных методов разработки ПО в Google, который вы как раз и найдёте здесь.

2. Разработка ПО

2.1. Репозиторий исходного кода

> Большая часть кода Google хранится в едином репозитории исходного кода и доступна всем разработчикам компании.

Стоит обратить внимание и на два проекта с открытым исходным кодом — браузер Chrome и ОС Android, которые хранятся в отдельных репозиториях. Доступ для чтения некоторых, особо важных с точки зрения безопасности, частей кода этих проектов тщательно ограничен, однако это скорее исключения — и большинство проектов Google хранится в общем репозитории.

На январь 2015 года объём памяти репозитория составлял 86 терабайт информации — миллиарды файлов. 9 млн из них включали в общей сложности 2 млрд строчек исходного кода, 35 млн сохранённых изменений при скорости в 40 тысяч изменений за рабочий день [18].

Доступ, позволяющий запись, ограничен: только указанные владельцы каждого поддерева репозитория могут утвердить внесённые в него изменения. Однако каждый без исключения программист может прочесть любую часть кода, написать новый, внести локальные изменения, протестировать их и после отправить на утверждение основным владельцам; и если владелец согласен с ними, то он подтверждает их сохранение.

В Google создана культура, которая способствует тому, чтобы инженеры исправляли ошибки вне зависимости от того, задействованы они в проекте или нет. Это не только придаёт программистам уверенности в себе, но и в значительной степени улучшает инфраструктуру для тех, кто её использует.

Большая часть разработки приходится на «верхушку репозитория», а не на ветки. Это позволяет выявить проблемы интеграции на раннем этапе и в значительной степени уменьшает объём последующей работы по слиянию сущностей. Это также упрощает и ускоряет процесс исправления уязвимостей.

Автоматизированные системы проводят регулярные тесты, часто после любого изменения файла в транзитивной зависимости, пусть это и не всегда необходимо и оправдано. Эти системы в течение нескольких минут автоматически присылают автору и рецензентам уведомление о любом изменении, которое не прошло тестирование.

Большинство команд делают статус своей разработки максимально заметным, устанавливая отображающие элементы или даже цветные строчки кода (зелёный означает, что разработка выполнена успешно и все тесты пройдены; красный — тестирование не пройдено; чёрный — повреждённый/ нерабочий код). Это стимулирует разработчиков писать именно «зелёный код». Многие крупные команды также имеют «копа по разработке», который следит за тем, чтобы все изменения проходили тесты, и указывает разработчикам на ошибки в коде с тем, чтобы они были исправлены, либо самостоятельно отменяет неблагоприятное изменение. (Обычно наиболее опытные члены команды берут на себя эту роль по очереди.) Стремление к «зелёному коду» «на верхушке» делает процесс разработки эффективнее, даже в действительно крупных командах.

Владение кодом. Каждое поддерево репозитория может иметь файл, в котором перечислены идентификаторы пользователей-владельцев. Подкаталоги также наследуют владельцев от родительского каталога, хотя эту функцию можно и деактивировать. Владельцы каждого поддерева имеют исключительный доступ к записи, что будет более детально раскрыто в разделе о просмотре кода ниже. Необходимо, чтобы у каждого поддерева было как минимум два владельца, хотя на деле их получается больше, особенно в международных командах. Бывает, что вся команда перечислена в списке владельцев. Изменения в поддерево может внести любой работник Google, но именно владельцы должны их подтвердить. Это гарантирует, что каждое изменение ещё раз будет проанализировано компетентным программистом.

Узнать больше о репозитории Google можно здесь [17, 18, 21], а о том, как другая крупная компания решает эту задачу — здесь [19].

2.2. Система сборки кода

Google использует распределённую систему сборки Blaze, которая отвечает за компиляцию и объединение программ, а также за проведение тестов. Она обеспечивает стандартные команды для сборки и тестирования программ, которые актуальны в рамках всего репозитория. Эти стандартные команды и их оптимизированное внедрение означает то, что каждый программист

Google **легко и быстро может протестировать любое ПО из репозитория**. Эта системность — ключевой фактор, который в значительной степени упрощает внесение изменений в ПО любым разработчиком за пределами его проекта.

Программисты создают файл сборки, который Blaze использует для построения ПО. Считается, что такие файлы, как библиотеки, программы и тесты, имеют детальные декларативные спецификации сборки, которые точно устанавливают для каждой из них имя, исходные файлы, а также библиотеки или другие элементы, на которых они основаны. Эти спецификации сборки состоят из разделов, называемых «правилами сборки кода», каждый из которых определяет такие понятия, как, например, «Библиотека C++ с файлами исходного кода, которые зависят от других библиотек», и уже система сборки кода размещает все правила в определённой последовательности шагов, а именно: сперва компиляцию и объединение каждого исходного файла, а затем ряд шагов по установлению связи и определению того, какой компилятор и флаг компиляции необходимо использовать.

В некоторых случаях, в частности, написанное на языке Go ПО собирает файлы, которые могут быть образованы (и скорректированы) автоматически, так как информация о зависимости в файлах сборки часто является отвлечённым понятием в тех же файлах исходного кода. Но тем не менее их формально регистрируют в репозиторий. Это гарантирует, что система сборки кода сможет быстро определить взаимозависимости, проанализировав лишь файлы процесса построения, а не файлы исходного кода. Данная схема также позволяет избежать излишней связанности между системой сборки кода и компиляторами или другими средствами анализа для всех остальных поддерживаемых систем программирования.

Система сборки кода использует инфраструктуру распределённых вычислений. Работа каждой сборки, как правило, **распределена среди сотен и даже тысяч систем.** Это позволяет создавать действительно масштабные программы быстро или, например, проводить большое количество тестов одновременно.

Индивидуальные шаги по сборке кода должны быть «герметичны» — основаны на исходных значениях и максимально независимы друг от друга. Требование корректно указывать взаимозависимости — следствие распространения сборки: только заявленные исходные значения направляются системе напрямую, и именно на их основе впоследствии осуществляется этап сборки. В результате, в основе системы сборки лежат истинные зависимости. Даже компиляторы, которые задействует сама система, воспринимаются в качестве исходных значений.

Индивидуальные системы сборки заранее определены. Как следствие, система сборки кода может отображать результаты в кэш-памяти. Программисты могут вернуть своё рабочее пространство к старому номеру изменения, перестроить и получить то же самое двоичное значение. Более того, доступ к этой буферной памяти может быть у нескольких пользователей. (Чтобы наладить этот процесс как следует, нам нужно было устранить неопределённость в инструментах, задействованных файлом сборки, например, очищая выходные данные от временных меток).

Система сборки надёжна. Она отслеживает взаимозависимости на основе правил сборки и знает, когда необходимо перестроить целевые значения, если действие, призванное их достичь, претерпело изменения (даже когда сами исходные значения остались прежними и, например, изменились лишь параметры компилятора). Это также связано с изменением файла сборки по ходу или изменением исходных файлов в процессе сборки: в таких случаях необходимо переформулировать заданную команду. Однако больше нет необходимости начинать всё заново.

Результаты сборки хранятся в «облаке». Они также включают промежуточные результаты. В случае, если другому запросу сборки необходимы те же результаты, система сборки будет автоматически использовать их снова, нежели перестраивать всё заново, даже если запрос исходит от другого пользователя.

Инкрементальные перепостроения происходят быстро. Система сборки постоянно находится в памяти с тем, чтобы постепенно анализировать все файлы с произошедшими изменениями.

Предварительная проверка. В Google есть специальные инструменты для автоматического проведения ряда тестов перед началом ревью кода кода и/или подготовки к утверждению изменения в репозитории. Каждое поддерево репозитория может содержать определённый файл настройки, который определяет, какие тесты необходимо провести и когда стоит это сделать: во время ревью кода, сразу после сохранения либо оба раза. Тесты могут проводиться синхронно: например, начинаться до отправки информации об изменении на проверку и/или до сохранения изменения в репозитории (подходит для коротких тестов); или же они могут быть асинхронными, то есть результаты будут отправлены на электронную почту для последующего обсуждения. [Существует единая цепочка электронных писем, где и происходит ревью кода; вся информация в этой цепочке будет отображаться в специальном веб-инструменте, используемом для ревью кода].

Сотрудники тель-авивского офиса Google. Фото: Itay Sikolski

2.3. Ревью кода

В Google существуют отличные веб-инструменты для ревью кода, которые интегрированы в электронную почту и позволяют авторам кода и тем, кто его непосредственно проверяет, одновременно сравнивать различия (включая разноцветные строки кода) и параллельно оставлять комментарии. Когда автор кода запрашивает проверку, рецензент получает уведомление об этом на

электронную почту, в котором содержится ссылка на веб-инструмент для анализа непосредственно этого изменения. Уведомления также рассылаются, когда рецензенты оставляют комментарии. Кроме того, используемые в компании инструменты рассылают уведомления, содержащие результаты автоматизированных тестов и статического анализа.

> Все изменения, внесённые в репозиторий исходного кода, ДОЛЖНЫ быть проверены как минимум ещё одним программистом. Помимо этого, если автор изменения не является одним из владельцев файла, тогда вдобавок к этому один из владельцев также должен взглянуть и подтвердить правки.

В отдельных случаях владелец поддерева может подтвердить срочное изменение в поддереве до того, как оно было кем-либо проверено, но имя рецензента в любом случае должно быть указано. Как автор изменения, так и рецензент будут получать напоминания о том, что изменение необходимо ещё раз проверить и подтвердить. В таких случаях любые последующие изменения необходимо вносить отдельно, ведь первоначальные уже были подтверждены.

В Google есть инструменты, которые автоматически предлагают кандидатуры рецензентов для того или иного внесённого изменения. Они основываются на информации о том, кто является владельцем и автором кода, в который были внесены изменения, а также на списке последних рецензентов и количестве уже направленных и пока не рассмотренных запросов на ревью кода тому или иному рецензенту. Хотя бы один из владельцев поддерева, в которые были внесены изменения, должен проанализировать и подтвердить их. В остальном автор полностью в праве выбрать подходящих, на его взгляд, рецензентов.

Проблема возникает тогда, когда рецензенты оттягивают ревью кода либо неохотно принимают любые изменения — это в значительной степени замедляет процесс разработки. Выбор рецензентов самим автором помогает ускорить процесс, так как он интуитивно избегает тех, кто противится любому изменению, либо направляет запрос касательно минимального изменения менее щепетильным рецензентам и, соответственно, запросы касательно масштабных изменений — более внимательным к деталям разработчикам.

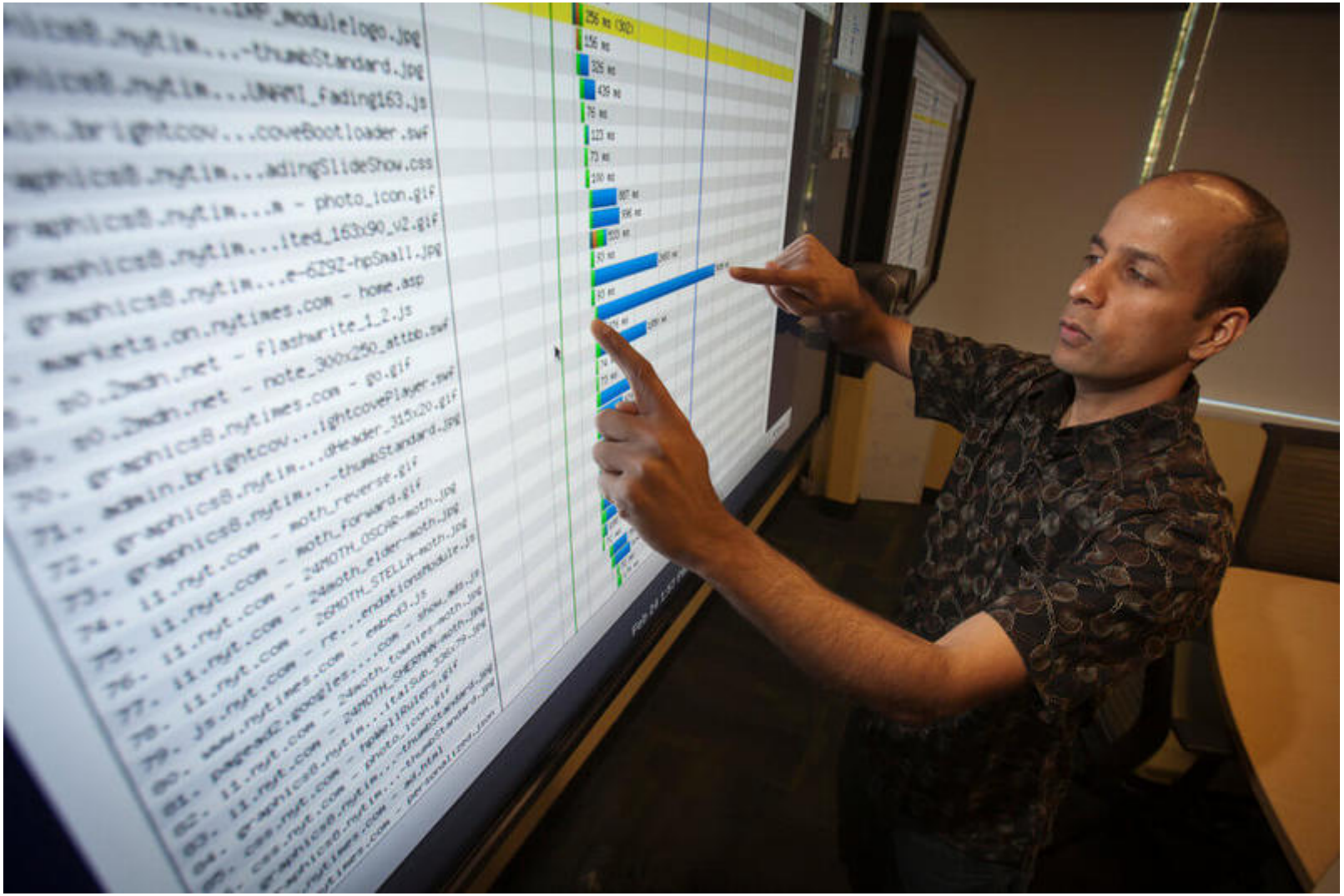
Любые письма с обсуждением ревью кода автоматически рассылаются всем разработчикам, задействованным в проекте. Любой из них вправе оставить комментарий касательно любого изменения (как до, так и после подтверждения

изменения), независимо от того, были ли они указаны в качестве рецензентов. Если обнаруживается ошибка, обычно ведётся поиск конкретного изменения, вызвавшего её, и разработчики указывают на это автору и другим рецензентам изменения.

Также можно рассылать запрос на ревью кода нескольким рецензентам одновременно и сохранять изменение после подтверждения одним из них (при условии, что автор либо рецензент являются непосредственными владельцами кода) даже до того, как другие рецензенты оставят какие-либо комментарии. Все комментарии, последующие за ревью, рассматриваются уже в процессе последующей доработки. Всё это в значительной степени ускоряет процессы.

Помимо основной части репозитория, существует также «экспериментальная» его часть, где стандарты подтверждения внесённых изменений не так строги. Однако код, который находится непосредственно в эксплуатации, должен храниться в основной части репозитория. Разработчиков призывают к тому, чтобы писать код именно в основной части (а не сперва — в экспериментальной и уже после переносить в основную), ведь эффективнее проводить ревью кода непосредственно в ходе процесса его написания. На практике программисты часто запрашивают возможность провести ревью написанного кода даже в экспериментальной части репозитория.

Инженерам рекомендуют вносить небольшие индивидуальные изменения, а также разбивать масштабные изменения на ряд мелких — чтобы рецензенту было удобнее проверять. Так и автору легче уследить, какие конкретно правки предлагает внести рецензент. Одним из вариантов ограничения масштаба изменения может служить определение и последующая отметка инструментами ревью количества строчек изменённого кода. Так, например, изменения в пределах 30-99 добавленных (удалённых, перемещённых) строчек отмечаются как «средние», более 300 строчек — «крупные», 1000-1999 — «нереально крупные» (freakin huge) и т.д. (В стиле Google — периодически заменять устоявшиеся понятия на забавные аналоги несколько раз в году. Однако в последнее время эта практика претерпела некоторые изменения. Самые последние инструменты анализа отмечают размер изменений как “S”, “M”, “L”, “XL»).



Тестирование скорости загрузки элементов сайта в главном офисе Google в Маунтин-Вью. Фото: Peter DaSilva for The New York Times

2.4. Тестирование

Тестирование компонентов крайне приветствуется и широко используется в Google. Подразумевается, что все части кода, который находится в эксплуатации, проходят компонентное тестирование, и инструмент ревью кода укажет, если вдруг исходные файлы были добавлены без предварительного прохождения соответствующих тестов. Рецензенты кода обычно требуют, чтобы

любое внесённое изменение, которое призвано расширить функционал, сопровождалось добавлением соответствующих тестов. Mock-объекты (делают возможным построение упрощённых компонентных тестов даже для кода, который основывается на крупных библиотеках) пользуются популярностью.

Тестирование взаимодействия компонентов системы и регрессивное тестирование также широко используются.

Как уже описывалось выше, тестирование может проводиться автоматически в рамках ревью кода и процесса его сохранения.

Google также автоматизровал инструменты оценки тестового охвата. Результаты интегрируются как часть произвольно задаваемого исходного кода программы.

Нагрузочное тестирование до непосредственного внедрения — обычная процедура в Google. Командам необходимо подготовить таблицу или график, демонстрирующий, как ключевые показатели — в особенности период ожидания и частота появления ошибок — меняются в зависимости от уровня входящих запросов.

2.5. Отслеживание ошибок

В Google используется система отслеживания ошибок Buganizer. Она способна отслеживать баги, запросы функций, проблемы, с которыми сталкиваются пользователи и другие процессы (выход окончательной версии или же подговительные работы). Ошибки ранжируются в иерархической структуре, по каждой из них назначается ответственный сотрудник и список людей, которым направляется скрытая копия письма. При отправке на анализ изменения исходного кода рекомендуется, чтобы разработчики присваивали имя изменению в соответствии с номером возникшей проблемы.

Часто (однако не всегда) команды в Google просматривают нерешённые вопросы в своих компонентах, расставляют их в порядке значимости и, где возможно, назначают ответственных. Некоторые команды имеют отдельного человека для сортировки ошибок и их приоритизации, некоторые делают это на общих собраниях. Многие команды в Google присваивают ошибкам метки для того, чтобы отмечать, какие из них уже были отсортированы и в какой из версий ошибка была устранена.

2.6. Языки программирования

> Инженерам ПО в Google рекомендуется писать на одном из пяти официально утверждённых языках программирования: **C++**, **Java**, **Python**, **Go** или **JavaScript**. Ограничение выбора языка расширяет возможности последующего использования кода другими программистами и способствует их взаимодействию.

Также для каждого языка действуют специальные руководства по стилю оформления, гарантирующие, что код во всей компании написан в одном стиле, в рамках одинаковой разметки страницы, используя одинаковые наименования и т.д.

Помимо этого в компании организуется специальный процесс обучения — для универсального понимания этих рекомендаций. Более опытные программисты, которые заботятся о читабельности кода, обучают других, как писать код доступно, как и когда уместно употреблять наименования, характерные для того или иного языка. Происходит они это в рамках длительного ревью кода. Обучение продолжается до тех пор, пока рецензент окончательно не убедится в том, что его подопечный действительно овладел правилами написания кода на конкретном языке программирования. Каждое изменение, которое вносит новый нетривиальный код, должно быть одобрено тем, кто уже подтвердил уровень владения соответствующим языком.

Вдобавок к вышеперечисленным языкам, для определённых целей используются соответствующие **специализированные предметно-ориентированные языки** (например, язык сборки используется для определения целей сборки и из зависимостей).

Взаимодействие между разными языками происходит в основном посредством протокола передачи структурированных данных (Protocol Buffers). Такие протоколы позволяют закодировать структурированные данные с возможностью подключения расширений. Они включают предметно-ориентированный язык для определения структурированных данных вместе с компилятором, который обрабатывает и преобразовывает код в языки C++, Java, Python для последующего построения, доступа, сериализации и десериализации этих объектов. Google-версия таких протоколов интегрирована в RPC-библиотеку Google (Remote Procedure Call — удалённый вызов процедур), что делает возможным межъязыковой вызов удалённых процедур. RPC-технология осуществляет автоматическую сериализацию и десериализацию всех запросов и ответов на них.

Унификация — ключ к максимальному упрощению процесса разработки, даже в случае действительно крупной базы кода и разнообразия языков. В Google существует ряд команд, выполняющих все базовые операции по разработке ПО: извлечение, редактирование, сборка, тестирование, анализ, подтверждение, отчёт об ошибке и т.д. Все эти команды одинаковы для любого проекта или языка. Разработчикам не нужно изучать новый процесс разработки только потому, что редактируемый ими код оказывается частью чужого проекта или написан на другом языке.

2.7. Инструменты для отладки и профилирования

Серверы Google связаны с библиотеками, которые предоставляют инструменты для отладки на запущенных серверах. В случае сбоя на сервере, обработчик сигнала автоматически добавляет отслеживание стека в системный журнал, одновременно сохраняя файл ядра. Если сбой происходит в связи с недостатком динамической памяти, сервер сохраняет отслеживание стека выборки подмножеств активных динамических объектов распределяемых сайтов. Существуют также веб-интерфейсы для устранения ошибок, которые позволяют проанализировать входящие и исходящие RPCs (включая временные характеристики, частоту появления ошибок, ограничение скорости и т.д.), изменение флагового значения командной строки (например, увеличение словесного наполнения при регистрации определённого модуля), потребление ресурсов, профилирование и многое другое.

Эти инструменты в значительной степени упрощают процесс отладки до уровня, когда редко приходится прибегать даже к таким традиционным методам, как gdb (переносимый отладчик проекта GNU, способный производить отладку многих языков программирования).

2.8. Релиз-разработка

Внутри некоторых команд в Google определены релиз-инженеры, однако в основном эту роль выполняют инженеры-программисты.

Релизы происходят часто: еженедельный/ ежемесячный релиз — это стандарт. Некоторые команды обновляют продукты каждый день. Это возможно благодаря **автоматизации большинства стандартных релизных задач разработчика**. Такой регулярный выпуск обновлений и продуктов мотивирует программистов (намного сложнее вкладываться во что-то, что будет запущено через

несколько месяцев или даже лет) и поддерживает общую динамичность в компании, позволяет большее число итераций и, как следствие, больше возможностей для получения обратной связи и ответа на неё в определённые сроки.

Как правило, создание новой версии начинается в новой рабочей среде, синхронизируясь с «самыми свежими» строчками «зелёного кода» (т.е. к последнему изменению, которое удачно прошло тесты) и создавая новую ветку. Релиз-инженер может особым образом выделить дополнительные изменения, то есть объединить части основной и релизной веток. После этого программа дорабатывается с самого начала, проводятся тесты. Если какие-то из них проваливаются, вносятся дополнительные изменения — после чего они вновь объединяются с новой ветвью. Затем программа снова дорабатывается, проводятся новые тесты. Если тесты проходят успешно, исполняемые файлы и файлы данных обновляются. Все эти шаги автоматизированы, чтобы релиз-инженеру оставалось лишь задать некоторые простые команды и после выбрать, какие изменения стоит объединить с основными.

После обновления потенциальные файлы сборки обычно загружаются на **вспомогательный сервер** для дальнейшего **интеграционного тестирования небольшим числом пользователей** (иногда только командой разработчиков).

Очень полезно отправить копию всех запросов рабочего трафика не только на вспомогательный, но и на рабочий сервер для непосредственной обработки. Ответы от вспомогательного сервера удаляются, а от реального направляются обратно пользователям. Это гарантирует, что любые неполадки, способные вызвать впоследствии большие проблемы (например, сбой сервера), будут обнаружены до введения сервера в эксплуатацию.

Следующий шаг — развернуть один или больше **«канареечных» серверов,** которые обработают **определённое подмножество активного трафика**. В отличие от «вспомогательных» серверов, они обрабатывают и отвечают реальным пользователям.

После релиз распространяется на все серверы во всех центрах сбора данных. Такой постепенный выпуск релиза в течение нескольких дней осуществляется с тем, чтобы уменьшить влияние возможных сбоев из-за обнаруженных ошибок (которые не удалось зафиксировать на предыдущих этапах процесса) на проекты с высоким трафиком. Чтобы больше узнать о релиз-разработке е в Google, см. [7, глава 8], [15].

2.9. Подтверждение запуска

Запуск любого видимого пользователю изменения требует одобрения целого ряда людей вне команды разработчиков, которые впоследствии его внедряют. В частности, подтверждения (часто подвергаются тщательному ревью) требуются для того, чтобы гарантировать соответствие кода юридическим нормам, требованиям конфиденциальности, безопасности, надёжности (например, соответствующий автоматический контроль для обнаружения сбоев сервера и автоматического уведомления ответственных разработчиков), бизнес-требованиям и т.д.

Процесс запуска организован таким образом, чтобы соответствующие сотрудники всегда своевременно узнавали о том, что запускается новый продукт или функционал.

> В Google есть внутренний инструмент утверждения готовности продукта к запуску, который отслеживает все рецензии и положительные оценки, а также гарантирует соответствие запуска продукта определённому для него процессу. Этот инструмент легко настроить под индивидуальные требования каждого нового продукта.

Чтобы больше узнать о процессах запуска в Google, см. [7, глава 27].

2.10. Анализ неудач

При значительных сбоях или даже минимальном происшествии в продакшн-системе все причастные должны подготовить документ, в котором будут проанализированы ошибки. Документ описывает произошедший инцидент, включая название, краткое изложение, хронологию событий, основные причины, что сработало (не сработало) и перечень предпринятых мер.

> Основной фокус — на проблемах и на том, как их в будущем избежать, но ни в коем случае не на людях или распределении вины между ними.

В разделе о последствиях пытаются оценить масштаб проблемы: длительность сбоя, количество потерянных запросов (или провалившихся RCPs и т.д.) и убыток от неё. Хронология отображает всю цепочку событий, которые привели к сбою и шаги,

предпринятые дабы его устранить. В разделе «что сработало (не сработало)» описываются вынесенные уроки: какие методы помогли быстро обнаружить и устранить проблему, что пошло не так и какие конкретно действия (в идеале, конкретные ошибки и ответственные за них люди) необходимо предпринять, чтобы уменьшить вероятность повторения такой проблемы в будущем.

Больше информации — [7, глава 15].

2.11. Частые доработки

> Большинство ПО в Google переписывают каждые несколько лет.

Может показаться, что это очень затратно. Так и есть, на это уходит большая доля ресурсов Google. Однако подобная тактика также приносит значительные преимущества, которые являются ключевыми в вопросах динамичности компании и её долгосрочного успеха.

За последние годы требования к продуктам в значительной степени изменились, ПО-среда и все смежные с ней технологии тоже. Всё это в значительной степени влияет на потребности, желания и ожидания потребителей.

ПО, которому уже несколько лет, было разработано вокруг старого набора требований и, как правило, не всегда может отвечать текущим потребностям. Более того, оно является более громоздким и сложным в использовании. Переписывание кода избавляет его от чрезмерной сложности, элементов, которые отвечали уже не актуальным требованиям. Вдобавок, это позволяет передавать новым членами команды знания и чувство причастности. Последнее очень важно для стимулирования продуктивности работников: программисты вкладывают больше сил в разработку функционала и устранение ошибок в коде, который они воспринимают «своим».

Частое обновление кода также способствует мобильности программистов между несколькими проектами, что, в свою очередь, вызывает обмен идеями между разными командами. Оно также гарантирует, что код написан с использованием современных технологий и методов.

Сотрудники Google в Ирвине, Калифорния. Фото: officelovin.com

3. Управление проектами

3.1. 20% времени

> Разработчики могут посвящать 20 процентов офисного времени работе над любым проектом (без разрешения менеджера). Такое доверие к инженерам очень важно по нескольким причинам.

Во-первых, это позволяет любому человеку с хорошей идеей — пусть даже другие не посчитают её сразу таковой — получить достаточное время для разработки прототипа, демо-версии — и представить всю её ценность.

Во-вторых, это даёт менеджерам наглядное представление о том, чем занят их сотрудник. В других компаниях, где нет политики «20 процентов», инженеры часто работают над сторонними проектами, не информируя об этом менеджмент. Намного лучше, если они могут свободно делиться идеями, описывать прогресс в регулярных обновлениях о состоянии дел, даже если менеджмент не считает проект стоящим. Однако благодаря официальной политике компании и корпоративной культуре Google это возможно.

В-третьих, возможность заниматься проектами «по душе» подпитывает энтузиазм инженеров к работе, предотвращает выгорание на рабочем месте, вероятность которого намного выше, если сотрудник тратит 100 процентов рабочего времени на выполнение монотонных и однотипных задач. Разница в уровнях продуктивности между заинтересованными и мотивированными разработчиками по сравнению со «сгоревшими на работе» составляет около 20 процентов. Когда инженеры видят, что их коллеги вовлечены в сторонние экспериментальные проекты, это также мотивирует их попробовать свои силы в реализации новых идей.

3.2. Цели и ключевые результаты (OKRs)

Google требует от каждого сотрудника и команд в целом документировать цели и оценивать прогресс на пути их достижения. Команды ставят цели поквартально и ежегодно, формулируя их таким образом, чтобы впоследствии можно было провести количественную оценку. Это происходит на всех уровнях менеджмента в компании и, в конце концов, выливается в общую для всех цель.

Задачи для сотрудников и малых команд должны идти в ногу с целями более высокого уровня и более крупных команд — и быть неотъемлемой частью общих целей компании. В конце каждого квартала фиксируется прогресс и выставляется оценка: от 0,0 (нет прогресса) до 1,0 (100-процентное выполнение) по каждой задаче. Показатели OKRs каждой команды, как правило, доступны всем в Google (с некоторыми исключениями, если речь идёт о конфиденциальной информации), однако они не используются для оценки каждого сотрудника в отдельности.

> Цели должны быть высокими: необходимый уровень среднестатистической оценки — 65%. Это означает, что команда должна установить на 50% больше целей, чем она скорее всего достигнет на деле. Если команда сможет достичь значительно выше 65%, рекомендуется, чтобы цели на следующий квартал были ещё более амбициозными (и наоборот).

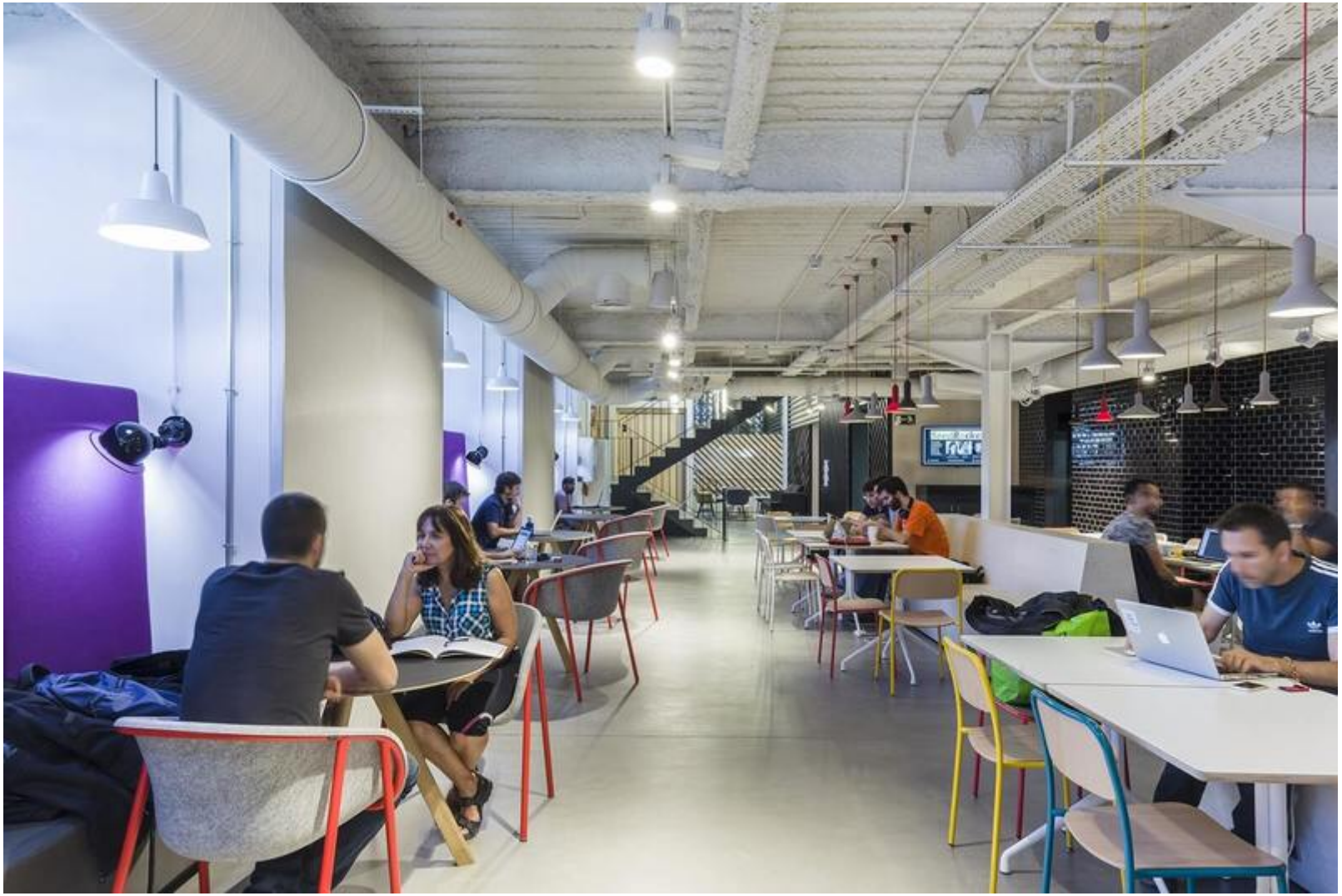
OKRs обеспечивают механизм общения для всей компании, ведь так каждый знает, над чем работает тот или иной отдел. Это мотивирует выполнять работу действительно хорошо, ведь разработчики знают о том, что их ждёт внутреннее собрание, на котором будут выставлены OKRs. Людьми движет внутреннее желание достичь высоких показателей, пусть напрямую те и не оказывают влияния на оценку их личностной работы или уровень заработной платы.

Определение максимально объективных и измеримых ключевых результатов гарантирует, чтобы естественное человеческое стремление преуспеть имеет прямой позитивный эффект на общее дело всей компании.

3.3. Утверждение проекта

> Несмотря на определённый процесс запуска продуктов, в Google не существует чёткого алгоритма выбора или отказа от того или иного проекта. Я работаю в компании уже более 10 лет, стал менеджером, но всё ещё не до конца понимаю, как мы принимаем такие решения.

Всё потому, что в рамках самой компании этот процесс может варьироваться. Менеджеры на каждом уровне ответственны за то, над какими проектами трудятся их команды, действуя по собственному усмотрению. В некоторых случаях это означает, что такие решения принимаются «снизу», то есть инженерам даётся право выбора, над какими проектами им хотелось бы работать в рамках своей команды. Бывают случаи, когда решение принимается сверху, и менеджеры выбирают, какой проект будет начат, в какой будут вложены дополнительные ресурсы, а какой пришло время закрыть.



Зона для совместной работы в мадридском офисе компании. Фото: officelovin.com

3.4. Корпоративные реорганизации

Иногда случается так, что принимается решение закрыть тот или иной проект. Тогда большое число работавших над ним инженеров вынуждены искать новые проекты в других командах.

Аналогичным образом иногда имеют место и «процессы дефрагментации», когда проекты, разбитые на несколько географических локаций, ограничиваются до нескольких стран. В таких случаях инженеру предоставляется право выбрать новую команду и роль из ряда доступных в определённой стране либо остаться в старой команде и проекте, но переехать в другую страну.

Помимо этого, часто происходят слияния или разделения команд, вносятся изменения в организацию процесса отчётности. Сложно сказать, насколько это действительно отличительная черта Google либо всё же подобная практика встречается и в других крупных компаниях. В целом, в технологически ориентированной компании регулярная реорганизация скорее на пользу, ведь позволяет избежать организационные недостатки — технологии и требования постоянно меняются.

4. Управление персоналом

4.1. Должностные функции

Как будет описано более детально ниже, Google разделяет карьерные лестницы для менеджмента и инженеров. Технический руководитель проекта и менеджер проекта — это две разные позиции. Компания внедряет исследования в процесс разработки, сопровождая инженеров менеджерами по продукту, проект-менеджерами и системными администраторами. Мы уверены, что именно эти методы и привели к развитию нынешней инновационной культуры в Google.

В компании есть некоторое число позиций в рамках самих процессов разработки. Каждая роль имеет чётко определённые перспективы роста (что влечёт за собой соответствующее увеличение зарплаты и т.д.), которые призваны поощрять хорошую работу сотрудника.

Основные позиции включают:

Менеджер разработки (Engineering Manager)

Это единственная позиция в списке, непосредственно связанная с управлением людьми. В рамках других позиций, как, например, инженер-программист (Software Engineer), менеджмент сотрудников также может иметь место — а вот менеджер разработки всегда занимается этим.

Часто эту позицию занимают бывшие инженеры-программисты, имеющие глубокие технические знания и хорошо развитые навыки межличностного общения.

> Существует разница между техническим лидерством и управлением людьми.

Менеджер разработки не обязательно ведёт проекты. Этим занимается техлид (Tech Lead), который может быть менеджером, но чаще всего это инженер-программист. За техлидом остаётся последнее слово во всех технических вопросах по проекту.

Менеджер разработки отвечает за выбор подходящего технического директора и за результаты работы своей команды. Он консультирует по рабочим вопросам и относительно продвижения по карьерной лестнице, оценивает работу сотрудников (используя обратную связь коллег каждого сотрудника), а также отвечает за финансовые поощрения. Engineering Manager также вовлечён в процесс найма сотрудников.

Руководители отделов могут управлять от 3 до 30 человек, хотя чаще всего это число составляет 8-12.

Инженер-программист (Software Engineer)

Большинство сотрудников, разрабатывающих ПО в Google, занимают эту позицию. Требования к ней в Google очень высоки. На работу берут лишь самых талантливых, ведь большинство проблем с ПО, которые так широко распространены в других компаниях, здесь отсутствуют либо минимизированы.

> В Google есть собственная система продвижения по карьерной лестнице для инженеров и менеджмента. Хотя теоретически инженер-программист может управлять коллегами либо передать эту обязанность менеджеру разработки, как таковое управление персоналом не является основным условием для продвижения, даже на самом высоком уровне.

На более высоких уровнях навыки лидерства действительно важны, но не всегда под ними подразумевается то, о чём мы привыкли думать. Например, создание действительно серьёзного продукта, который имеет значительное влияние или используется большим количеством разработчиков, достаточно. Это важно, потому что у действительно талантливых инженеров появляются хорошие карьерные перспективы. И для этого им необязательно брать на себя, возможно, не такую желанную функцию управления персоналом.
Подобный подход также позволяет избежать появления на менеджерских позициях людей, чьи межличностные навыки недостаточно развиты.

Учёный-исследователь (Research Scientist)

Требования к этой позицию крайне серьёзны и планка очень высока: необходимо продемонстрировать исключительные исследовательские способности, подкреплённые списком публикаций *и* способностью писать код.

> Большое количество действительно талантливых людей из университетской среды, кто с лёгкостью подошёл бы на роль инженера-программиста, не пройдут на роль учёного-исследователя в Google. Большинство докторов наук в Google — это скорее программисты, нежели исследователи.

Учёных-исследователей прежде всего оценивают по их непосредственному вкладу в исследования, включая список публикаций. Однако применимо к Google это описание среднестатистического программиста. Что та, что другая роль подразумевает проведение исследований, научные публикации, развитие новых идей для продуктов и новых технологий и, точно также, оба из них могут писать код. Учёный-исследователь в Google работает напрямую с программистами в рамках команды и параллельно проводит исследование в контексте тематики проекта. Практика внедрения исследований в процесс разработки вносит большой вклад в развитие продукта.

Системный администратор (Site Reliability Manager, SRE)

Поддержание работы операционных систем осуществляют команды инженеров-программистов, а не системные администраторы в их традиционном понимании. Предъявляемые к позиции SRE требования касательно написания кода могут быть ниже, чем к

программистам, если при этом компенсируются знаниями сетевой архитектуры или «внутренностей» Unix. Получить более детальную информацию по этой позиции можно здесь [7].

Менеджер по продукту (Product Manager)

Менеджеры по продукту отвечают за управление продуктом. Они являются своеобразными сторонниками пользователя, координируют работу разработчиков, продвигая функционал, который важен непосредственно для пользователя, кооперируются с другими командами, отслеживают ошибки и сроки, а также гарантируют, что на выходе получится продукт высокого качества. Менеджеры по продукту, как правило, не пишут код, но тесно работают с разработчиками, контролируя, чтобы те писали его должным образом.

Руководитель проекта / Технический руководитель проекта (Program Manager / Technical Program Manager)

Роль руководителя проекта схожа с позицией менеджера по продукту, однако человек на этой должности скорее занимается развитием не самого продукта, а управлением процессами и внутренними проектами. Технический руководитель проекта (Technical Program Manager) — опять же, что-то очень схожее, но требует специфических технических знаний, связанных с рабочими процессами (например, владение необходимой лингвистикой для работы с речевыми данными).

> Соотношение инженеров-программистов и менеджеров по продукту (руководителей проекта) варьируется между 4 к 1 и 30 к 1.

Место для отдыха в дублинском офисе Google. Фото: officelovin.com

4.2. Обустройство офиса

Google широко известен неформальным подходом к обустройству офиса: горки, бассейны с шариками и игровые комнаты. Всё это помогает привлечь и удерживать талантливые кадры. Отличные бесплатные кафетерии помогают достичь этой цели и ненавязчиво намекают сотрудникам, чтобы те задержались в офисе подольше. Таким образом, голод никогда не является

причиной ухода с работы. Часто также устанавливаются «мини-кухни», где работники могут перекусить и обменяться идеями в неформальной обстановке, ведь очень часто разговоры начинаются именно здесь. Спортивный зал, занятия спортом, массаж — всё это поддерживает работников в отличной форме и прекрасном настроении, способствуя повышению продуктивности и удержанию кадров.

> Рабочие места расположены согласно свободной планировке, часто довольно плотно. Хотя выгода от этого и неоднозначна [20], но такая планировка точно способствует общению, пусть иногда и ценой концентрации разработчиков. Также выходит достаточно экономно.

Каждый работник имеет индивидуальное рабочее место, однако очень часто они меняются (примерно каждые 6-12 месяцев, как правило, в результате расширения команды). Обычно менеджеры определяют рассадку, дабы максимально способствовать комфортному общению, ведь всегда легче обратиться к тому, кто сидит поблизости.

Помещения Google также оснащены переговорными комнатами с ультрасовременным оборудованием для проведения видеоконференций: присоединиться к запланированному разговору можно, лишь подтвердив приглашение в календаре.

4.3. Обучение

Google способствует обучению персонала несколькими способами:

- Новые сотрудники (New Googlers, Nooglers) проходят обязательный вводный учебный курс;
- Инженерно-технический персонал (Technical staff) начинает с курса «Codelabs»: краткие онлайн-курсы, каждый из которых включает ряд упражнений по написанию кода и рассчитан на освоение определённой технологии;
- Google предлагает работникам на выбор как онлайн-, так и офлайн-курсы.
- Компания также поддерживает инициативу сотрудников, желающих обучаться во внешних учреждениях.

В дополнение к обязательному вводному курсу, каждый «нуглер» получает в помощь ментора (Mentor) и коллегу (Buddy), которые помогают поскорее влиться в коллектив и корпоративную культуру в целом. Негласно менторство также происходит в рамках

частых встреч с менеджером, командой, во время ревью кода, дизайна и других неформальных процессов.

Фото: roi4my.com

4.4. Ротация персонала

Перемещения между различными командами приветствуются, ведь это способствует обмену знаниями и технологиями в компании и улучшает коммуникацию внутри команд.

Трансферы между проектами и/или офисами доступны для каждого работника после года работы на одной позиции. Также рекомендуется, чтобы инженеры-программисты периодически выполняли задачи в других частях организации, например, в рамках шестимесячного перемещения на другую позицию.

4.5. Оценка работы персонала и вознаграждения

В Google крайне приветствуется обратная связь. Разработчики могут давать друг другу позитивный фидбэк посредством *peer bonuses* и *kudos*.

Система «peer bonuses» подразумевает то, что любой работник может два раза в год номинировать коллегу на получение стодолларовой премии — за выполнение не только своих прямых обязанностей. Для этого нужно просто заполнить веб-форму и обосновать причину. Обычно о вручении такого бонуса сообщается на общем собрании команды.

Члены команды также могут выражать уважение другу другу посредством официально оформленных писем — *kudos*, что обеспечивает публично выраженное признание хорошей работы друг друга, но не подразумевает финансового поощрения. Для *kudos* необязательно, чтобы человек выходил за рамки своих прямых обязанностей и здесь нет ограничений на то, какое количество раз человек может его получить.

Менеджеры также могут назначать бонусы, включая премии, например, по завершении проекта. И так же, как и во многих других компаниях, в Google работники получают ежегодный бонус по результатам работы или вознаграждения в виде участия в доле капитала компании.

> Процесс продвижения по карьерной лестнице в Google чётко определён. Он включает номинирование человека на должность как самостоятельно, так и со стороны менеджера, самоанализ, анализ работы коллегами, оценку работы менеджером. Конкретные решения принимаются комитетом по

> продвижению на основе всей полученной информации. Гарантия того, что действительно достойные люди получают повышение, важна для грамотной мотивации остальных сотрудников.

Проблема неудовлетворительных показателей работы, напротив, решается путём обратной связи от менеджера и, если это необходимо, с внедрением плана по улучшению показателей, который подразумевает постановку чётких задач и оценку прогресса по ходу их выполнения. Если всё это не работает, тогда последует увольнение, но это очень редкая практика в Google.

Для оценки работы менеджера его подчинённые два раза в год заполняют анонимные опросники. Все результаты агрегируются и после передаются менеджеру. Такая система восходящей обратной связи очень важна для поддержания и улучшения работы менеджеров в рамках всей компании.

5. Вывод

Мы очень кратко описали большинство ключевых методов разработки ПО, применяемых в Google. Конечно, сейчас это крупная и разнонаправленная компания, некоторые части которой организованы по-своему. Однако описанным здесь практикам всё-таки следует большинство команд Google.

> Учитывая множество применяемых в Google методов разработки ПО и другие причины успеха компании, порой очень сложно проследить объективную взаимосвязь между применением индивидуальных практик и общим результатом. Однако эти практики прошли в Google проверку временем, где стали предметом коллективного, пусть и субъективного, суждения тысяч талантливейших разработчиков.

Для тех, кто пытается убедить коллег использовать в своих организациях подобную систему, возможно, веским аргументом будет упомянуть, что она «достаточно хороша для Google».

Ссылки

[1] Build in the Cloud: Accessing Source Code, Nathan York.
[2] Build in the Cloud: How the Build System works, Christian Kemper.
[3] Build in the Cloud: Distributing Build Steps, Nathan York.
[4] Build in the Cloud: Distributing Build Outputs, Milos Besta, Yevgeniy Miretskiy and Jeff Cox.
[5] Testing at the speed and scale of Google, Pooja Gupta, Mark Ivey, and John Penix, Google engineering tools blog, June 2011.
[6] Building Software at Google Scale Tech Talk, Michael Barnathan, Greg Estren, Pepper Lebeck-Jone, Google tech talk.
[7] Site Reliability Engineering , Betsy Beyer, Chris Jones, Jennifer Petoff, Niall Richard Murphy, O'Reilly Media, April 2016, ISBN 978-1-4919-2909-4.
[8] How Google Works, Eric Schmidt, Jonathan Rosenberg.
[9] What would Google Do?: Reverse-Engineering the Fastest Growing Company in the History of the World , Jeff Jarvis, Harper Business, 2011.
[10] The Search: How Google and Its Rivals Rewrote the Rules of Business and Transformed Our Culture , John Battelle, 8 September 2005.
[11] The Google Story , David A. Vise, Pan Books, 2008.
[12] Searching for Build Debt: Experiences Managing Technical Debt at Google, J. David Morgenthaler, Misha Gridnev, Raluca Sauciuc, and Sanjay Bhansali.
[13] Development at the speed and scale of Google , A. Kumar, December 2010, presentation, QCon.
[14] How Google Tests Software, J. A. Whittaker, J. Arbon, and J. Carollo, Addison-Wesley, 2012.
[15] Release Engineering Practices and Pitfalls, H. K. Wright and D. E. Perry, in Proceedings of the 34th International Conference on Software Engineering (ICSE ’12), IEEE, 2012, pp. 1281–1284.
[16] Large-Scale Automated Refactoring Using ClangMR, H. K. Wright, D. Jasper, M. Klimek, C. Carruth, Z. Wan, in Proceedings of the 29th International Conference on Software Maintenance (ICSM ’13), IEEE, 2013, pp. 548–551.
[17] Why Google Stores Billions of Lines of Code in a Single Repository, Rachel Potvin, presentation.
[18] The Motivation for a Monolithic Codebase, Rachel Potvin, Josh Levenberg, to be published in Communications of the ACM, July 2016.
[19] Scaling Mercurial at Facebook, Durham Goode, Siddharth P. Agarwa, Facebook blog post, January 7th, 2014.
[20] Why We (Still) Believe In Private Offices, David Fullerton, Stack Overflow blog post, January 16th, 2015.
[21] Continuous Integration at Google Scale, John Micco, presentation, EclipseCon, 2013.