

Джеймс
Уиттакер

Джейсон
Арбон

Джефф
Карено

КАК ТЕСТИРУЮТ В GOOGLE



How Google Tests Software

James Whittaker
Jason Arbon
Jeff Carollo

▲Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Джеймс
Уиттакер

Джейсон
Арбон

Джефф
Каролло

КАК ТЕСТИРУЮТ В GOOGLE

 ПИТЕР® ИНОУЛ

ББК 32.973.2-018-07

УДК 004.415

У13

Дж. Уиттакер, Дж. Арбон, Дж. Каролло

У13 Как тестируют в Google. — СПб.: Питер, 2014. — 320 с.: ил.

ISBN 978-5-496-00893-8

В книге описано тестирование программных продуктов в Google: как устроены процессы, как организованы команды, какие техники используются, кто ответственен за качество. Принципы, на которых построено тестирование в Google, применимы в проектах и компаниях любого размера. Авторы книги сами работали над продуктами Google, создавая инструменты тестирования, настраивая процессы и занимаясь непосредственно тестированием. Книга рассчитана на профессионалов из индустрии разработки программного обеспечения: специалистов по тестированию, программистов, менеджеров.

12+ (Для детей старше 12 лет. В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018-07

УДК 004.415

Authorized translation from the English language edition, entitled How Google Tests Software; ISBN 9780321803023; by James A. Whittaker, Jason Arbon, Jeff Carollo; published by Pearson Education, Inc Copyright © 2012 Pearson Education, Inc

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Права на издание получены по соглашению с Pearson Education, Inc. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-0321803023 англ.

© Pearson Education, Inc., 2012

ISBN 978-5-496-00893-8

© Перевод и издание на русском языке ООО Издательство «Питер», 2014

© Рецензирование ООО «Иннова Системс», 2014

© Оформление ООО Издательство «Питер», ООО «Иннова Системс», 2014

Содержание

Предисловие к русскому изданию.....	10
Вступление от Альберто Савоя.....	12
Вступление от Патрика Коупленда	16
Предисловие	22
Пара слов о книге	24
Благодарности	25
Об авторах	27
Глава 1. Первое знакомство с организацией тестирования в Google.....	28
Качество ≠ Тестирование	33
Роли	34
Организационная структура	37
Ползти, идти, бежать.....	38
Виды тестов	40
Глава 2. Разработчик в тестировании.....	44
Жизнь разработчика в тестировании.....	46
Как организованы процессы разработки и тестирования	47
Кто такие разработчики в тестировании на самом деле?	51
Ранняя стадия проекта	52
Структура команды.....	54
Проектная документация	55

Интерфейсы и протоколы.....	57
Планирование автоматизации.....	58
Тестируемость	59
Пример работы разработчика в тестировании.....	63
Выполнение тестов	74
Определения размеров тестов.....	75
Как мы используем размеры тестов в общей инфраструктуре.....	77
Преимущества разных размеров тестов	79
Требования к выполнению тестов.....	81
Тест-сертификация	87
Интервью с основателями программы тест-сертификации	89
Как мы собеседуем на позицию разработчиков в тестировании.....	96
Интервью с разработчиком инструментов Тедом Мао	103
Интервью с создателем WebDriver Саймоном Стюартом	105
Глава 3. Кто такой инженер по тестированию	109
Тестирование, обращенное к пользователю	109
Инженер по тестированию	110
Планирование тестирования.....	114
А — значит Attribute	117
С — значит Component	121
С — значит Capability	122
Риск.....	132
Анализ рисков.....	133
Снижение рисков	138
Напоследок о рисках	141
Пишем тест-кейсы	145
Интересные факты из жизни багов.....	150
Немного подробнее о Buganizer	151
Как мы нанимаем инженеров по тестированию	164
Собеседование с инженерами по тестированию	168
Управление тестированием в Google	173
Тестирование в режиме сопровождения.....	178
Эксперимент с Quality Bots.....	182
Эксперимент BITE.....	195
Регистрируем баги с BITE	197
Просмотр багов в BITE	199
Запись и воспроизведение сценариев в BITE	201
Ручные и исследовательские тесты в BITE	205
Уровни BITE	205
Google Test Analytics.....	206

Бесплатное тестирование	212
Внешние тестировщики.....	216
Интервью с инженером по тестированию Google Docs Линдси Уэбстер	218
Интервью с инженером по тестированию YouTube Эппл Чоу.....	224
Глава 4. Тест-менеджер	231
Кто такой тест-менеджер	231
Жонглирование людьми и дирижирование проектами	233
Влияние	235
Интервью с Анкитом Мехтой, тест-менеджером Gmail	237
Интервью с Хуном Даном, тест-менеджером Android.....	244
Интервью с Джоэлом Хиноски, тест-менеджером Chrome	249
Директор по тестированию	254
Интервью с Шелтоном Маром, директором по тестированию проектов Search и Geo.....	255
Интервью с директором разработки инженерных инструментов Ашишем Кумаром	259
Интервью с Суджаем Сани, директором по тестированию в индийском Google.....	263
Интервью с тест-менеджером Брэдом Грином.....	268
Интервью с Джеймсом Уиттакером	272
Глава 5. Как мы улучшали тестирование в Google	279
Роковые ошибки в процессе тестирования Google	279
Будущее разработчика в тестировании	282
Куда движется роль инженера по тестированию	283
Что станет с тест-директором и тест-менеджером.....	285
Будущее инфраструктуры тестирования	285
В завершение	286
Приложение А. Тест-план для Chrome OS	287
Обзор тем	287
Анализ рисков	288
Непрерывное тестирование каждой сборки.....	289
Ежедневное тестирование лучших сборок.....	289
Тестирование перед выпуском	290
Ручное и автоматизированное тестирование.....	290
Разработка и качество тестов	291
Каналы выпуска	291
Обратная связь.....	291
Репозитории тест-кейсов.....	292
Панели мониторинга тестов.....	292
Виртуализация	292

Производительность	293
Нагрузочное тестирование, продолжительное тестирование	
и тестирование стабильности	293
Фреймворк выполнения тестов Autotest	293
Производители железа.....	293
Лаборатория проверки оборудования	294
Фермы для сквозных автотестов	294
Тестирование AppManager в браузере.....	294
Тестируемость браузера.....	295
Оборудование.....	296
График	296
Ключевые моменты тестирования.....	298
Необходимые документы и ресурсы	299
Приложение Б. Тестовые туры для Chrome	300
Тур покупателя.....	300
Тур студента	301
Рекомендуемые области для тестирования.....	302
Тур международных звонков	302
Рекомендуемые области для тестирования.....	302
Тур ориентиров	303
Рекомендуемые ориентиры для Chrome	303
Тур «не спим всю ночь»	303
Рекомендуемые области для тестирования.....	304
Тур предпринимателя.....	304
Инструменты в Chrome	305
Тур неблагополучных районов.....	305
Неблагополучные районы в Chrome OS	305
Тур персонализации	306
Способы настройки Chrome	306
Приложение В. Посты из блога об инструментах и коде	307
Охотимся на баги и потерянное время вместе с BITE	307
QualityBots идет в атаку	310
RPF: Record Playback Framework	312
Google Test Analytics — теперь с открытым кодом	315
Полнота	315
Скорость	315
Действенность.....	316
Польза.....	316

Всем тестировщикам из Google, Microsoft и всех остальных компаний, которые заставили меня мыслить нестандартно.

— Джеймс А. Уиттакер

Моей жене Хизер и моим детям Луке, Матео, Данте и Одессе, которые думали, что я все это время работал в Starbucks.

— Джейсон Арбон

Маме, папе, Лорен и Алексу.

— Джейфф Каролло

Предисловие к русскому изданию

Я пришла в тестирование в 2006 году маленьким тестировщиком на большой аутсорсный проект. Сначала я научилась тестировать, заводить баги и общаться с разработчиками и менеджерами. Со временем я стала писать тесты, научилась планировать и управлять тестированием. У меня появилась своя команда.

И чем дальше, тем больше мне становилось понятно, что тестировщики только находят проблемы, но не могут их исправить. Они не могут сделать так, чтобы проблема больше не повторилась. И я чувствовала, что тестирование может приносить больше пользы.

Я начала ездить на конференции, читала книги и статьи по тестированию, общалась с коллегами по индустрии. Везде учили, как лучше тестировать, как находить больше ошибок, как быстрее находить ошибки. Тестировщики не хотели выходить за рамки своей профессии. Им как будто нравилось чувствовать собственную важность от того, что они нашли много багов.

Ответы на свои вопросы я нашла в статьях и докладах Джеймса Уиттакера. Его основная идея в том, что тестирование должно перестать просто предоставлять информацию и начать влиять на качество. Главная задача тестирования, говорил Уиттакер, — это уменьшение количества ошибок в процессах разработки. Тогда улучшится качество выпускаемого продукта.

Создать процесс, в котором сложно допустить ошибку, — вот настоящая цель тестирования. Мы не можем полностью избавиться от ошибок, но можем построить работу так, что сделать сразу правильно будет легче, чем ошибиться.

В Google пошли именно в эту сторону, отказавшись от тестирования, которое просто сообщало об ошибках. «Служба тестирования» трансформировалась в «Направление продуктивности разработки», которое помогает разработчикам и менеджерам делать меньше ошибок и получать обратную связь как можно раньше. Тестировщики в Google влияют на качество, потому что встраивают его на всех этапах разработки программных продуктов.

Книга «Как тестируют в Google» рассказывает, как тестиировщики в Google пришли к пониманию, что нужно меняться, как строили новые процессы, каких результатов достигли и как видят дальнейшее развитие тестирования. Очень здорово, что компания Google настолько открыта и делится своим опытом. Но все же, я думаю, что именно приход Уиттакера в Google послужил катализатором процесса написания этой книги. Слишком уж много у него энергии и желания делиться знаниями с внешним миром.

Я начала переписываться с Уиттакером в 2009 году, а в марте 2010-го познакомилась с ним лично на конференции Swiss Testing Day. Он оказался очень простым в общении. С ним можно было обсудить тренды мирового тестирования и посмеяться над тестиировщиками-консерваторами. Джеймс очень остроумный и харизматичный оратор. В 2011-м я слушала его мастер-класс «Как тестируют в Google» на конференции EuroSTAR. Он легко парировал реплики из зала вроде «в авиационном софте не получится сделать так, как в Google» простой фразой «так вы даже не пробовали».

Эта книга повторяет легкий и остроумный стиль общения Джеймса. Вы почувствуете характер Уиттакера, читая ее. Мы очень старались сохранить в переводе эту легкость и живость. Я хотела, чтобы текст получился таким, как будто его написал кто-то очень знакомый: наш коллега, который говорит на том же языке, что и мы. В итоге книга получилась не похожей на большинство технических книг на русском языке, переведенных сухо и формально.

Если вы начинающий тестиировщик — прочитайте книгу сейчас, а потом еще раз через год. Если вы опытный тестиировщик — дайте ее почитать вашим разработчикам и менеджерам. Если они не загорятся идеей сделать тестирование «как в Google», задумайтесь, стоит ли с ними работать. Если вы менеджер или разработчик — прочитайте и сравните с тем, как работает тестирование в вашей компании.

Пожалуйста, будьте осторожны. Не нужно следовать советам ребят из Google вслепую: то, как у них реализованы процессы, лишь частный случай. Вычлените из книги главное: тестирование как выделенная проверка не работает, тестировать должен каждый участник проекта, качество нужно встраивать на все уровни разработки. Если применить эти принципы в вашей компании, скорее всего, вы получите совершенно другую реализацию.

Думайте, «как в Google», а не делайте «как в Google». Приятного чтения!

Спасибо всем ребятам, кто помогал в начале проекта: Илье Фомину, Ане Якшиной, Наташе Курашкиной, Леше Лянгузову, Севе Лотошникову, Игорю Любину, Ване Федорову, Жене Ткаченко, Илье Шубкину. Спасибо «Иннове» и Геворку, что этот проект состоялся. Спасибо Ксюше Развенской и Сергею Сурганову, что помогали вычитывать финальный вариант.

Юля Нечаева

P.S. У меня остался только один вопрос к Джеймсу Уиттакеру: о каком инциденте с костюмом Бедняжки Мэри говорит Фред в четвертой главе?

Вступление от Альберто Савоя

Писать вступление к книге, автором которой вы сами хотели быть, — сомнительное удовольствие. Все равно что стать шафером на свадьбе вашего друга и девушки, на которой *вы сами* давно мечтали жениться. Но Джеймс Уиттакер — хитрый малый. Прежде чем спросить, не соглашусь ли я написать это предисловие, он воспользовался моей слабостью к мексиканской кухне и угостил меня превосходным обедом. Только после того, как мы опустошили пару бутылок пива, он задал мне этот вопрос «на десерт». К этому моменту я был такой же мягкий и податливый, как порция гуacamоле, которую только что прикончил. *«Si, señor»* — вот и все, что я смог ответить. План сработал, и вот Джеймс стоит со своей книгой, как с невестой, а я собираюсь произнести свадебную речь.

Как я уже сказал, он хитрый малый.

Ну, поехали... предисловие к книге, которую я хотел бы написать сам. Звучит трогательная свадебная музыка.

Нужна ли миру еще одна книга по тестированию ПО? Особенно если это еще одна книга по тестированию ПО, написанная плодовитым Джеймсом Уиттакером, которого я зову «октомамой»¹ книг по тестированию? Разве мало книг, раздающих сомнительные и устаревшие советы по методологии тестирования? Да, таких книг достаточно, но это произведение, боюсь, к ним не относится. Вот почему я хотел бы написать его сам. Миру действительно нужна именно такая книга по тестированию.

Интернет значительно изменил подход к проектированию, разработке и распространению программных продуктов. Многие передовые методы тестирования,

¹ Не поняли, о чём речь? Погуглите!

описанные в книгах прошлых лет, сейчас уже неэффективны, иногда вообще бесполезны, а в некоторых случаях даже вредны. В нашей отрасли условия меняются очень быстро. Книги, написанные пару лет назад, уже похожи на древние медицинские манускрипты по лечению пиявками и сверлению дырок в черепе для изгнания злых духов. Такие книги лучше переработать в подгузники для взрослых, чтобы они не попадали в руки доверчивых неофитов.

Развитие отрасли разработки ПО идет семимильными шагами, и я не удивлюсь, если через десять лет эту книгу тоже можно будет пустить на подгузники. Но пока не произошла очередная смена парадигмы, книга «Как тестируют в Google» дает очень своевременный и практичный взгляд на то, как одна из самых успешных и быстро развивающихся интернет-компаний справляется с уникальными задачами тестирования в XXI веке. Джеймс Уиттакер и его соавторы ухватили самую суть того, как Google успешно тестирует одни из самых сложных и популярных программных продуктов нашего времени. Я наблюдал становление отрасли и знаю, о чем говорю.

В 2001 году я присоединился к Google в качестве директора по разработке. Тогда у нас было около двухсот разработчиков и... целых *три* тестировщика! Мои разработчики уже тестировали свой код сами, но метод TDD (Test-Driven Development, разработка через тестирование¹) и средства автоматизации тестирования (такие как JUnit) только делали свои первые шаги. Поэтому наше тестирование было по большей части случайным и зависело от добросовестности разработчика, написавшего код. Но это нормальная ситуация для стартапа. Нам нужно было двигаться быстро и не бояться рисковать, иначе мы бы не смогли конкурировать с материальными игроками нашей отрасли.

Однако компания росла, и наши продукты становились все более значимыми для пользователей и клиентов. Возьмем хотя бы AdWords: один из сервисов, за которые я отвечал, он быстро превращался в основной источник монетизации веб-сайтов. Пришло время пересмотреть свой подход к тестированию и увеличить инвестиции в эту область. Имея в арсенале только трех тестировщиков, мы не нашли других вариантов, кроме как попытаться вовлечь в тестирование разработчиков. Я и еще несколько сотрудников Google продвигали, развивали и обучали юнит-тестированию. Мы агитировали разработчиков начать писать тесты и использовать хотя бы JUnit для их автоматизации. Переход шел со скриптом. Идея тестирования собственного кода не вызывала у разработчиков ажиотажа. Для стимулирования сотрудников каждую пятницу на наших пивных вечеринках с говорящим назвлением «Thanks God, It's Friday!» я награждал разработчиков, которые писали тесты. Я напоминал дрессировщика, который дает собачкам лакомство за выполненный трюк, но, по крайней мере, это привлекало внимание

1 Разработка через тестирование – процесс разработки программного обеспечения, который основан на том, что тесты пишутся до того, как будет написан код. Таким образом, функциональные обязанности кода определяются заранее. – Примеч. перев.

ние к тестированию. Неужели мне повезло, и я смог заманить разработчиков в тестирование конфетками?

К сожалению, нет. Идея с поощрением не сработала. Разработчики поняли, что для создания нормальных тестов им нужно писать две-три строки кода юнит-тестов на каждую строку тестируемого кода. Вдобавок тесты тоже нужно сопровождать, и в них самих могут заводиться баги. Стало очевидно, что одними разработчиками в юнит-тестировании сыт не будешь. Нам нужны были интеграционные тесты, системные тесты, тесты пользовательского интерфейса и прочие полезные штуки. Нам предстояло еще много узнать о тестировании и многому научиться, причем сделать это надо было быстро! Очень быстро!

К чему такая спешка? Я не верю, что даже самое изощренное тестирование может привести к успеху изначально неудачную идею или непродуманный продукт. Но я верю, что плохое тестирование может загубить хороший продукт, компанию или, по меньшей мере, сможет замедлить ее рост и предоставит фору конкурентам. Это как раз портрет Google того времени. Мы были на пути к успеху, и только проблемы в тестировании удерживали компанию от решающего шага в светлое будущее. Нужно было выработать верные стратегии тестирования, адекватные нашим сверхзвуковым скоростям роста числа пользователей, продуктов и сотрудников. Мы вовсю применяли инновационные решения, нестандартные подходы и уникальные инструменты — нам нельзя было замедляться. Конечно, не все работало. Но в процессе мы получили ценные уроки и научились практикам, которые теперь может применить любая компания, желающая развиваться со скоростью Google. Мы поняли, как сохранить баланс между качеством продукта и скоростью разработки, не теряя ни в чем. Эта книга рассказывает о процессе, к которому мы пришли, о том, какие идеи лежат в его основе, и почему он работает. Если вы хотите понять, как Google справляется с проблемами тестирования в XXI веке, в среде современных интернет-, мобильных и клиентских приложений — вы обратились по адресу. Конечно, я хотел бы оказаться тем человеком, который рассказал бы вам все это, но Джеймс и его соавторы уже сделали это за меня. Главное, что им удалось передать суть тестирования в Google.

И последняя ремарка: Джеймс Уиттакер — человек, благодаря которому эта книга написана. Он пришел в Google, проникся нашей культурой, взялся за большие и важные проекты, выпустил такие громкие продукты, как браузер Chrome, операционная система Chrome OS и десятки других поменьше. В какой-то момент он стал лицом тестирования в Google. В отличие от его других книг, в этой — большая часть материала не его авторства, он скорее выступил в роли корреспондента с места события и представил свой репортаж об эволюции тестирования ПО в Google. Помните об этом, читая книгу, потому что Джеймс наверняка постарается прибрать всю славу себе!

Пока население Google росло с 200 до 20 000 сотрудников, было немало людей, которые оставили свой след в разработке и благодаря которым наши идеи тести-

рования живут. Джеймс благодарит многих из них. Они появляются на страницах книги в отступлениях и интервью. Тем не менее ни я, ни Джеймс не сделали так много, как Патрик Коупленд — архитектор нашей текущей организационной структуры и руководитель направления продуктивности разработки Google. Все тестировщики Google подотчетны Патрику. Это его видение реализовалось в наших подходах и вот теперь описано в этой книге Джеймсом. Если есть человек, благодаря которому тестирование ПО в Google организовано именно так, то это Патрик... Я говорю это не просто потому, что он мой начальник. Я говорю это потому, что он мой начальник, *a еще* он приказал мне это сказать!

Альберто Савоя — директор по разработке и популяризатор инноваций в Google. Он пришел в компанию в 2001 году и возглавил выпуск Google AdWords, а кроме того, сыграл ключевую роль в развитии культуры разработки и юнит-тестирования в компании. Он написал книгу «The Way of Testivus» и раздел «Beautiful Tests» в книге издательства O'Reilly «Beautiful Code».

Примечание от Джеймса Уиттакера: Согласен на все сто! Я ничего не придумывал, просто пересказал схему организации, которую создал Патрик. И я говорю это не потому, что он разрешил мне написать эту книгу. Как мой начальник, он *заставил* меня написать ее!

Вступление от Патрика Коупленда

Мое приключение в Google началось в марте 2005 года. Если вы прочитали вступление Альберто, то вы уже примерно представляете, что тогда происходило в компании. Google был еще маленький, но в нем уже начали проявляться признаки болезни роста. Это было время стремительных технологических изменений: на замену клиент-серверной модели приходили динамический контент и облачные технологии.

В первую неделю я, как и остальные новички, сидел в фирменной трехцветной кепке с пропеллером, слушая, как основатели компании обсуждают корпоративную стратегию на еженедельном собрании «Thanks God, It's Friday». Тогда я еще не понимал, во что влип. Я был наивен настолько, чтобы испытывать энтузиазм, и достаточно опытен, чтобы начать подозревать неладное. Мой предыдущий опыт с пятилетними циклами разработки продукта выглядел не очень убедительно по сравнению со скоростью роста и масштабами Google. Что еще хуже, я был единственным тестировщиком в кепке новичка. Я надеялся, что где-то есть и другие!

Когда я пришел в Google, там было не больше тысячи инженеров. В команду тестирования входили 50 штатных сотрудников и несколько временных, которых я никогда не мог запомнить. Все это называлось «Службой тестирования». Она занималась тестированием пользовательского интерфейса и прыгала с проекта на проект по мере надобности. Как можно догадаться, это была не самая популярная команда Google.

На тот момент этого было достаточно. Основными направлениями Google были поиск и реклама. Сфера деятельности Google тогда была значительно уже, чем сей-

час, и для поддержки качества достаточно было исследовательского тестирования. Но мир менялся. Пользователи приходили в веб в невиданных количествах, и веб стал ориентироваться на приложения, а не на документы. Все чувствовали необходимость роста и изменений. Если ты не хотел оказаться за бортом индустрии, нужно было уметь оперативно масштабировать продукт и быстро выпускать его на рынок.

Внутри Google команда службы тестирования сражалась с драконами масштаба и сложности. Решения, которые хорошо работали в мелких однородных проектах, теперь только обжигали наших славных тестировщиков, прыгающих с одного горящего проекта на другой. Прибавьте к этому стремление Google форсировать выпуск продуктов. Нужно было что-то делать. Я должен был выбрать между масштабированием ручного труда и полным изменением правил игры в тестирование. Тестирование должно было радикально измениться, чтобы соответствовать радикальным изменениям в индустрии и в Google.

Мне бы очень хотелось сказать, что я воспользовался своим огромным опытом и с ходу выдал идеальный механизм организации тестирования. Но, честно говоря, мой опыт научил меня только тому, как делать *не надо*. Каждая структура тестирования, в которой я участвовал (или которую возглавлял), была по-своему ущербной. В ней всегда что-то работало плохо: иногда код, иногда тесты, иногда сама команда. Я отлично знал, как можно оказаться погребенным под грудой технических и качественных проблем, когда каждая новая идея с ходу отвергается, если она хоть немного угрожает хрупкому проекту. Если я чему-то и научился к этому времени, так это тому, как *не нужно* проводить тестирование.

Я уяснил одно: Google уважает компьютерные технологии и искусство программирования. Если тестировщики хотят вливаться в эту компанию, они должны быть технически грамотными и уметь писать код. Только так остальные инженеры будут воспринимать тестировщиков равными себе.

Ну и уж если я собрался что-то менять в структуре тестирования Google, то начинать надо было с самих тестировщиков. Я представлял идеальную команду и то, как она будет обеспечивать качество, и все время возвращался к одному очень важному условию: команда может сделать качественный продукт только в том случае, если за качество отвечают все ее участники. Менеджеры, разработчики, тестировщики... все. Мне казалось, что лучший способ это обеспечить — превратить *тестирование* в фичу *кода*. Фича «тестирование» должна была обладать теми же правами, что и любая другая фича, которую может увидеть пользователь. А для создания фич нам нужен разработчик.

Нанять тестировщиков, которые умеют программировать, нелегко, а найти разработчиков, которые умеют тестировать, еще сложнее. Но текущее состояние дел было хуже некуда, и я решил действовать. Я хотел, чтобы тестировщики могли сделать больше для своих продуктов, и в то же время я хотел изменить природу тестирования и разделить ответственность за него. Для этого мне нужна была под-

держка команд разработки. Я еще ни разу не встречал подобной организационной структуры за все время работы в отрасли, но был уверен, что она подойдет Google, и считал, что наша компания к ней готова.

К сожалению, мало кто в компании разделял мой энтузиазм по поводу столь принципиальных и фундаментальных изменений. Рассказывая о своем подходе «все будет точно так же, только по-другому», я обнаружил, что мне стало труднее найти компанию для обеда! Разработчиков, казалось, пугала мысль о том, чтобы играть более важную роль в процессе: «А тестировщики нам зачем?» Тестировщики тоже не горели желанием менять привычный стиль работы. Попытки что-либо изменить сталкивались с сильным сопротивлением.

Я продолжал гнуть свою линию, так как боялся, что инженерная работа Google погрязнет в технических долгах и проблемах качества, а мне придется вернуться к пятилетним циклам разработки, которые я с радостью оставил в доисторическом клиент-серверном мире. Google — компания гениев, ратующих за инновации, и этот дух изобретательства просто несовместим с долгими циклами выпуска. Игра стоила свеч. Я убедил себя, что однажды эти гении прочувствуют необходимость объединения практик разработки и тестирования в один циклический и высокотехнологичный (как на фабрике) процесс и перейдут на мою сторону. Они поймут, что мы уже не стартап. Наша стремительно растущая пользовательская база, набирающий скорость снежный ком багов и плохо структурированный код могут разрушить уютный мирок разработчиков.

Я начал обходить команды разработки продуктов, разъясняя свою позицию, и старался подбирать подходящие аргументы для каждого. Разработчикам я рисовал картину непрерывной сборки, быстрого развертывания и процесса разработки настолько стремительного, что останется время на эксперименты. Общаюсь с тестировщиками, я упирал на их желание стать равноправными разработчиками инженерами с равным уровнем мастерства, равным вкладом в продукт и равным уровнем оплаты.

Разработчики считали, что если уж мы собирались нанимать людей, которые могут писать функциональность, то пусть они это и делают. Некоторые так сильно противились моей идеи, что почтовый ящик моего начальника был всегда полон советов, как лучше приструнить мое безумие. К счастью, мой начальник не внял этим рекомендациям.

Тестировщики, к моему удивлению, реагировали аналогично. Они привыкли к текущему состоянию дел, охотно оплакивали свой статус, но не торопились его менять.

Когда я пожаловался своему начальнику, он сказал: «Это Google. Если ты хочешь, чтобы что-то было сделано, — сделай это».

Сказано — сделано. У меня было достаточно единомышленников, чтобы стартовать конвейер собеседований, и мы начали беседовать с кандидатами. Задача была не из легких. Мы искали людей, обладающих умениями разработчика

и менталитетом тестировщика. Нам были нужны специалисты, которые умели программировать и хотели применить свои навыки в разработке инфраструктуры, инструментов и автоматизации тестирования. Мы должны были придумать новые критерии набора персонала и проведения собеседований, а затем объяснить этот процесс нашим рекрутерам, которых вполне устраивал текущий ход дел.

Первые несколько месяцев были самыми тяжелыми. В процессе проверки часто отбраковывались неплохие кандидаты. Возможно, они недостаточно быстро решали какую-нибудь хитроумную задачу по программированию или плохо проявляли себя в области, важной для интервьюера, но на самом деле не имеющей никакого отношения к навыкам тестирования. Я знал, что с наймом у нас будут проблемы, и каждую неделю строчил докладную за докладной. Они уходили Ларри Пейджу, который был (и остается) последней инстанцией в процессе найма. Он утвердил достаточное количество кандидатов, и моя команда начала расти. Я иногда думаю, что мое имя у Ларри ассоциируется только с фразой «Нам нужны тестировщики!».

Конечно, я поднял столько шума, пытаясь найти поддержку, что пути назад у нас уже не было. За нами наблюдала вся компания, и неудача могла стать фатальной. От маленькой тестовой команды с вечно меняющимся штатом контрактников ожидали великих свершений. И все же — на фоне моих битв с наймом и сокращения числа внештатных сотрудников — я начал замечать изменения. Чем сильнее становился дефицит тестировщиков, тем больше тестовой работы приходилось выполнять разработчикам. Многие команды приняли вызов. Думаю, если бы технологии не развивались, уже этого было бы достаточно, чтобы со временем выйти на нужный уровень.

Но технологии не стояли на месте, поэтому правила разработки и тестирования быстро менялись. Эпоха статического веб-контента уходила в прошлое. Автоматизация не успевала за и без того отстающими браузерами. Взваливать тестирование на разработчиков в то время, когда они столкнулись с серьезными технологическими сдвигами, было неразумно. Мы не могли организовать нормальное тестирование этих приложений даже в ручном режиме, не говоря уже об автоматизации.

Команды разработчиков испытывали сильное давление. Google начал покупать компании со своими веб-приложениями (YouTube, Google Docs и прочие), которые только усиливали нагрузку на нашу внутреннюю инфраструктуру. Проблемы, с которыми я сталкивался в тестировании, не уступали проблемам, с которыми сталкивались разработчики при написании кода. Я пытался решить проблему тестирования, которую просто невозможно было решить отдельно от разработки. Рассматривать тестирование и разработку как обособленные дисциплины или даже как разнородные задачи было в корне неверно, и мы не могли решить ни одну из проблем. Улучшение команды тестирования стало бы всего лишь временным успехом.

Но все-таки положительные сдвиги начались. Полезно нанимать умных людей — они могут сами исправить ситуацию. К 2007 году позиции дисциплины

тестирования укрепились. Мы неплохо справлялись с завершающей фазой цикла выпуска. Команды разработки знали, что могут рассчитывать на нас как на партнеров в производстве. Правда, наше существование как команды поддержки позднего цикла ограничивало нас рамками традиционной модели контроля качества. Несмотря на неплохие результаты, мы все еще не достигли состояния, в котором я хотел бы оказаться. Я разобрался с проблемой найма, тестирование двигалось в нужном направлении, но мы все еще слишком поздно включались в процесс.

Наша концепция «Тест-сертификации» делала успехи (вы прочитаете про нее дальше в книге). Мы общались с командами разработчиков и помогали улучшить чистоту кода и внедрить юнит-тестирование на ранней стадии. Мы разрабатывали инструменты и обучали команды основам непрерывной интеграции, чтобы наши продукты всегда были готовы к тестированию. Бесчисленные мелкие усовершенствования и приемы, многие из которых описаны в книге, развеяли скептицизм. Тем не менее чего-то не хватало. Разработка все еще была разработкой, а тестирование так и оставалось тестированием. Все ингредиенты для изменения культуры присутствовали, но нам был нужен катализатор, который смог бы запустить объединение дисциплин.

Глядя на организацию, которая выросла из моей идеи о привлечении разработчиков на позиции тестирования, я понял, что тестирование было лишь частью нашей работы. У нас были команды, которые создавали все инструменты — от репозиториев исходного кода до инфраструктуры для багтрекинговой системы. Среди нас были инженеры по тестированию, инженеры по выпуску продуктов, разработчики инструментов и консультанты. Меня поражало, как сильно тот аспект нашей работы, который не являлся непосредственно тестированием, влиял на производительность. Возможно, наша команда называлась «службой тестирования», но наши обязанности были намного шире.

Поэтому я решил оформить все официально и сменил название команды на «направление продуктивности разработки». Смена названия привела к культурному сдвигу. Люди начали говорить не о контроле качества и тестировании, а о производительности. Повышение производительности — наша работа, а тестирование и контроль качества — работа всех, кто пишет код. Это означает, что и тестирование, и контроль качества входят в обязанности разработчика. Команда продуктивности разработки должна заботиться о том, чтобы разработчик мог заниматься этими двумя вещами.

На первых порах идея была абстрактной, а наш девиз «Ускоряем Google» понапачу звучал наигранно. Со временем мы выполнили обещание. Наши инструменты ускорили темпы разработки, мы переходили от одного бутылочного горлышка к другому, расчищали пути и решали проблемы, с которыми сталкивались разработчики. Наши инструменты помогли разработчикам писать тесты и наблюдать их результаты при каждой сборке. Тестовые сценарии уже не выполнялись локально на компьютере тестировщика. Их результаты публиковались на информационных

панелях и собирались от версии к версии, так что в итоге они становились частью общедоступных данных о готовности приложения к выпуску. Мы не просто требовали участия разработчиков, мы упрощали его. Разница между тестированием и обеспечением продуктивности наконец-то стала ощутимой: Google мог создавать новое с меньшими затратами и обходиться без накопления технического долга.

Результаты? Пожалуй, я не буду предвосхищать содержание книги, потому что ее написали для того, чтобы рассказать о них миру. Авторы приложили огромные усилия, чтобы на основании собственной работы и опыта своих коллег из Google свести наш секретный рецепт успеха к конкретному набору практических методов. Мы добились успеха во многих отношениях — от сокращения времени сборки до автоматизированного тестирования в формате «запустил и забыл» и публикации в открытом доступе инструментов тестирования. Когда я писал это вступление, в направлении продуктивности разработки работало около 1200 специалистов, что немногим больше численности всего технического персонала в 2005 году, когда я пришел в Google. Бренд «продуктивности» укрепился, а наше стремление ускорить Google стало неотъемлемой частью нашей технической культуры. С того первого дня, когда я сидел озадаченный и неуверенный на пятничной встрече, наша команда проделала путь длиной во много световых лет. Единственное, что не изменилось с тех пор, — моя трехцветная кепка с пропеллером. Она лежит у меня на столе и напоминает о том, как далеко мы ушли.

Патрик Коупленд — старший директор направления продуктивности разработки, вершина пирамиды тестирования в Google. Все тестировщики в компании подчиняются Патрику, а его руководителем, кстати, является Ларри Пейдж, со-учредитель и исполнительный директор Google. Карьере Патрика в Google предшествовала почти десятилетняя работа в Microsoft на должности директора по тестированию. Он часто выступает публично и известен как архитектор используемой в Google технологии быстрой разработки, тестирования и развертывания ПО.

Предисловие

Разрабатывать программные продукты сложно. Тестирувать эти продукты тоже сложно. Когда кто-то говорит о разработке и тестировании в масштабах веб, он подразумевает Google. Если вы хотите узнать, как в одной из самых известных интернет-компаний решаются проблемы тестирования в грандиозных рамках всей Сети, то вы держите в руках правильную книгу.

В Google ежедневно тестируются и выпускаются сотни миллионов строк кода, распределенного по миллионам исходных файлов. Миллиарды операций по сборке запускают выполнение миллионов автоматизированных тестов в сотнях тысяч экземплярах браузеров ежедневно. Операционные системы строятся, тестируются и выпускаются в пределах одного календарного года. Браузеры собираются ежедневно. Веб-приложения выпускаются почти непрерывно. В 2011 году сто новых фич Google+ были выпущены за сто дней.

Таковы масштабы и скорость Google — они же масштабы развития веб. О том, как Google организовал тестирование в таких условиях, расскажет эта книга. Мы покажем, как проектировалась, внедрялась и сопровождалась эта инфраструктура. Мы познакомим вас с людьми, которые повлияли как на разработку основных концепций этой структуры, так и на ее реализацию.

Конечно, так было не всегда. Путь, который прошел Google к тому, где он сейчас, не менее интересен, чем технологии тестирования, которые мы используем. Давайте переведем стрелки часов на шесть лет назад — тогда ситуация в Google слабо отличалась от ситуации в других компаниях, с которыми мы работали: дисциплина тестирования была второстепенной.

Специалисты этой области получали меньше, хотя и работали больше. Тестирование в основном было ручным, а люди, которые могли автоматизировать процесс, быстро переходили в разработчики, где они могли «приносить больше

пользы». Основателям команды, которая сейчас называется направлением продуктивности разработки, пришлось прорываться сквозь заблуждения о тестировании и корпоративную культуру, ставившую героические авралы на работе выше повседневной инженерной рутине. Сегодня зарплата тестировщиков Google сопоставима зарплате разработчиков, их премии и темпы карьерного роста сравнялись. То, что культура тестирования пережила непростой рост компании (по всем параметрам — количеству, разнообразию продуктов и объемам продаж) и структурные реорганизации, должно придать уверенности компаниям, которые решат следовать примеру Google. Тестирование можно организовать правильно. Добиться вменяемого отношения к нему со стороны разработчиков и руководства — реально.

Сейчас все больше компаний делает ставку на веб, поэтому технологии тестирования и организационная структура, описанные в этой книге, могут получить широкое распространение. Если вы с нами — читайте, эта книга объяснит вам, как добраться до цели.

Материал книги по тестированию разделен на части с описанием отдельных ролей, причастных к тестированию. В первой главе мы рассмотрим все роли в общем и разберем концепции, процессы и тонкости контроля качества Google. Обязательно прочтите этот раздел.

Остальные главы можно читать в любом порядке. Сначала мы расскажем о роли разработчика в тестировании (Software Engineer in Test, SET), потому что с нее началось современное тестирование в Google. Разработчик в тестировании — тестировщик с высокой технической квалификацией, поэтому в этой главе много технических подробностей. Но мы постарались подать их в более общем ключе, чтобы основные концепции понял даже неподготовленный читатель. В следующей главе мы познакомимся с инженером по тестированию (Test Engineer, TE). Глава получилась большой, потому что обязанностей у инженера по тестированию много. В Google эти специалисты решают уйму разных задач на протяжении всего цикла разработки продукта. Эта роль хорошо знакома многим традиционным тестировщикам. Скорее всего, этот раздел книги будет максимально востребован, потому что он адресован самой широкой аудитории.

В книге мы соблюдали баланс между теоретическими основами и практическими советами ключевых участников, сыгравших важные роли в становлении тестирования или в истории ключевых продуктов Google. Эти интервью будут интересны всем, кто пытается опробовать в своей команде или компании процессы тестирования в стиле Google.

Остается последняя глава, которую не должен упускать ни один читатель, интересующийся темой. В ней Джеймс Уиттакер делится своими представлениями о том, как развивается тестирование в Google, и делает прогнозы о будущем тестирования в Google и в отрасли в целом. Думаю, многие читатели найдут этот материал поучительным, а может быть, даже в чем-то провокационным.

Об иллюстрациях

Из-за сложности рассматриваемых тем и графической природы интернет-контента некоторые иллюстрации из третьей главы дают лишь общее представление о концепциях. Они не предназначены для подробного рассмотрения. Если вы захотите просмотреть эти рисунки на своем компьютере, загрузите их по адресу www.informit.com/title/9780321803023.

Пара слов о книге

Когда Патрик Коупленд предложил мне написать эту книгу, я колебался. В конечном итоге мои сомнения подтвердились. Я боялся, что многие будут спрашивать, нет ли в Google более подходящего человека для этой работы, — и они спрашивали. Или толпы людей захотят участвовать в работе над книгой — и это тоже сбылось. Но главная причина была в том, что все мои предыдущие книги я писал для новичков. И серия «How to Break...», и моя книга «Exploratory Testing»¹ содержат законченные, цельные истории, у которых есть начало и конец. С этой книгой все иначе. Читатели, конечно, могут «проглотить» ее за один присест, но она задумана как справочник по решению реальных задач в Google, как больших, так и малых, из которых складывается наша практика тестирования.

Я думаю, что людям с опытом тестирования ПО в ИТ-компаниях книга принесет больше пользы, чем новичкам, потому что они могут сравнить процессы в Google с процессами, к которым они привыкли. Я представляю себе, как матерые тестировщики, менеджеры и директора берут книгу, находят интересующий их раздел и читают его, чтобы узнать, как нужная задача решена в Google. Но я-то привык к другому способу подачи материала!

В работе над книгой ко мне присоединились два соавтора, которые раньше не публиковались. Оба — первоклассные специалисты, проработавшие в Google дольше меня. Джейсон Арбон работает инженером по тестированию, хотя по сути он прирожденный организатор. Он здорово повлиял на реализацию многих идей и технических средств, описанных в главах этой книги. Пересечение наших карьерных путей изменило нас обоих.

Джефф Каролло — тестировщик, перешедший в разработчики. Безусловно, это лучший тестировщик-разработчик, которого я когда-либо встречал. Джефф

¹ Предыдущие книги Джеймса Уиттакера по тестированию «How to Break Software: A Practical Guide to Testing» («Как сломать программу: практическое пособие по тестированию») и «Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design» («Исследовательское тестирование: советы, хитрости, туры и техники тест-дизайна») на русский язык не переводились. — Примеч. перев.

один из немногих людей, которым удалось успешно организовать «полнейшую автоматизацию» (я имею в виду такую, которую можно запустить и забыть), — код тестов написан так хорошо и настолько автономен, что не требует участия автора. Они оба великолепные соавторы, и мы постарались, чтобы наши голоса в книге звучали в унисон.

Многие мои коллеги по Google помогли с материалом для книги. У каждой статьи-отступления есть свой автор. Мы подготовили для вас интервью с ключевыми сотрудниками Google, повлиявшими на организацию тестирования. Мы решили, что проще включить мнения людей, определивших процесс тестирования Google, прямо в книгу, чем перечислять тридцать соавторов. Конечно, не каждое интервью будет интересно всем, но они четко выделены в тексте, так что вы можете выбирать, читать или пролистывать заметку. Мы от всей души благодарим этих участников и заявляем: если нам где-то не удалось воздать должное их работе, то вся вина за это лежит на нас. Английский язык меркнет перед описанием их блестящей гениальности.

Приятного чтения, отличного тестирования — и желаем вам всегда находить (и исправлять) те баги, которые вы ищете!

Джеймс Читтакер

Джейсон Арбон

Джефф Каролло

Киркленд, штат Вашингтон

Благодарности

Мы благодарим всех технических специалистов Google, неустанно улучшающих качество продуктов. Мы восхищены открытой и распределенной культурой управления и организации технической работы в Google, которая позволила нам свободно исследовать и экспериментировать по ходу формирования практики и методологии тестирования.

Мы особо выделим следующих ребят, которые тратили свои силы и даже рисковали, чтобы привести тестирование к желаемому виду: Алексис О. Торрес, Джо Мухарски, Даниэль Дрю, Ричард Бустаманте, По Ху, Джим Рирдон, Теджас Шах, Джули Ральф, Эриэл Томас, Джо Михаил и Ибрагим Эль Фар.

Спасибо нашим редакторам, Крису Гузиковски и Крису Зану, стоячески переносившим наш технический жаргон.

Спасибо собеседникам, поделившимся своими взглядами и опытом в интервью: Анкиту Мехта, Джоэлу Хиноски, Линдсей Уэбстер, Эппл Чоу, Марку Стрибеку, Нилу Норвицу, Трейси Бялик, Руссу Руферу, Теду Мао, Шелтону Мару, Ашишу Кумару, Суджай Сани, Брэду Грину, Саймону Стюарту и Хунг Дан.

Отдельное спасибо Альберто Савоя, убедившему нас в преимуществах метода быстрого создания прототипа и итераций. Спасибо Google, персоналу кафетерия и шеф-поварам за отличную еду и кофе. Спасибо за благожелательные отзывы Филу Валигоре, Алану Пейджу и Майклу Бахману. И наконец, спасибо Пату Коупленду, который нас всех собрал и обеспечил финансирование такой энергичной и талантливой группы специалистов в области качества.

Об авторах

Джеймс Уиттакер — технический директор Google¹, ответственный за тестирование Chrome, Maps и веб-приложений Google. Перешел в компанию из Microsoft, имеет профессорскую степень. Джеймс — один из самых известных в мире специалистов в области тестирования.

Джейсон Арбон — инженер по тестированию в Google, ответственный за Google Desktop, Chrome и Chrome OS. Он выступал в роли руководителя разработки многих тестовых инструментов с открытым кодом и внутренних экспериментов по персонализации. До прихода в Google работал в Microsoft.

Джефф Карollo — разработчик в тестировании, ответственный за тестирование Google Voice, Toolbar, Chrome и Chrome OS. Он консультировал десятки групп разработки Google, помогая им повышать качество кода. В 2010 году перешел в разработчики, сейчас руководит разработкой Google+ API. До прихода в Google тоже работал в Microsoft.

¹ В начале 2012 года Джеймс Уиттакер перешел из Google в Microsoft. — Примеч. перев.

Первое знакомство с организацией тестирования в Google

Есть один вопрос, который я слышу чаще других. В какой бы стране я ни был, на какой бы конференции ни выступал, этот вопрос обязательно всплывает. Даже наши новобранцы задают его сразу же после прохождения вводного курса: «*Так как же Google тестирует ПО?*».

Не знаю, сколько раз я уже отвечал на этот вопрос и как много разных вариантов ответа дал. Мои ответы постоянно меняются — чем дольше я работаю в Google, тем больше узнаю всевозможных тонкостей нашего подхода к тестированию. Меня давно посещали мысли, что стоит зафиксировать свои знания на бумаге. Когда Альберто (который часто грозит переработать все книжки по тестированию в подгузники для взрослых, чтобы они принесли хоть какую-то пользу) предложил мне написать книгу, я понял, что никуда от этого не денусь.

И все-таки я хотел подождать. Во-первых, я не считал себя самым подходящим автором. Было много людей, которые работали в Google намного дольше меня, и я хотел дать им возможность первыми написать о своем опыте. Во-вторых, я был директором по тестированию Chrome и Chrome OS (кстати, сейчас эту должность занимает один из моих бывших подчиненных), поэтому видел организацию тестирования в Google только с одной стороны. Мне нужно было узнать еще много всего о тестировании в Google.

В Google тестирование ПО — часть централизованной системы, которую мы называем направлением продуктивности разработки. Она охватывает инструменты для разработки, тестирования (от юнит-уровня до исследовательского) и организации процессов выпуска программных продуктов. Мы создаем и поддерживаем множество общих инструментов и тестовую инфраструктуру для веб-проектов поиска, рекламной платформы, мобильных приложений, видеосервисов. Короче, для всего, что Google делает в интернете. В Google решены проблемы производительности и масштабирования, что позволяет нам, несмотря на огромные размеры компании, выпускать программы со скоростью стартапа. Как верно заметил Патрик Коупленд в предисловии к книге, этой магией мы во многом обязаны именно команде тестирования.

НА ЗАМЕТКУ

В Google тестирование ПО — часть централизованной системы, которую мы называем направлением продуктивности разработки.

В декабре 2010-го мы выпустили Chrome OS, и я передал руководство проектом одному из своих подчиненных, а сам погрузился в работу с другими продуктами. Тогда и началась эта книга. Мой первый пост в блоге «Как в Google тестируется ПО»¹ стал первой ласточкой, и с тех пор все закрутилось. Через полгода книга была готова, и я пожалел, что не начал писать ее раньше. За эти шесть месяцев я узнал о тестировании в Google больше, чем за два предыдущих года, а для новичков Google моя книга стала частью вводного курса.

Эта не единственная книга о том, как большие компании тестируют ПО. Я работал в Microsoft, когда Аллан Пейдж, Би-Джей Роллисон и Кен Джонстон написали книгу «How We Test Software at Microsoft», и сам применял все методы, описанные в книге. Компания Microsoft тогда была лидером в тестировании. Она подняла тестирование на один уровень с разработкой. Тестировщики Microsoft были самыми востребованными спикерами конференций. Первый директор по тестированию, Роджер Шерман, привлекал в Редмонд талантливых ребят со способностями к тестированию со всего мира. Это был золотой век тестирования.

Компания выпустила большую книгу, чтобы описать свой опыт.

Я не успел поработать над этой книгой, так как пришел в Microsoft поздно, но мне выпал второй шанс. Я попал в Google в тот самый момент, когда тестирование здесь было на подъеме. Команда направления продуктивности разработки стремительно выросла с пары сотен до 1200 человек. Проблемы роста, о которых Патрик говорит в своем предисловии, мы уже преодолели, и наше подразделение было на пике своего развития. Блог тестирования Google ежемесячно собирал сотни тысяч просмотров, а конференция GTAC² стала ведущей в индустрии тестирования ПО.

1 <http://googletesting.blogspot.com/2011/01/how-google-tests-software.html>

2 GTAC — Конференция Google по автоматизации тестирования (Google Test Automation Conference, www.GTAc.biz).

Вскоре после моего прихода Патрика повысили, и теперь ему подчинялось около дюжины директоров и технических менеджеров. Если у тестирования ПО и была вторая волна развития, то центром этой стихии был Google.

Это значит, что история тестирования в Google тоже заслуживала своей большой книги. Но вот незадача — я не пишу больших книг. Да и Google славится своим простым и лаконичным подходом к разработке. Надеюсь, что моя книга получилась такой же.

Книга рассказывает, что значит быть тестировщиком в Google, и описывает, как мы решаем проблемы, связанные с масштабом, сложностью и массовым использованием наших проектов. Здесь вы найдете информацию, которой больше нигде нет, но если и этого не хватит, чтобы удовлетворить ваше жгучее любопытство к процессу тестирования, то в сети можно узнать еще больше — просто погуглите.

Есть еще кое-что, и я должен это сказать. Скорее всего, организацию тестирования «как в Google» будут перенимать многие компании по мере того, как все больше программных продуктов переходит с десктопов в веб. Если вы читали книгу от Microsoft, то не думайте, что найдете здесь много общего с ней. У обеих книг по три автора, в каждой книге описана организация тестирования в крупной компании — разработчике ПО. На этом общее заканчивается. Трудно найти два подхода к тестированию, которые различались бы сильнее.

НА ЗАМЕТКУ

Скорее всего, организацию тестирования «как в Google» будут перенимать многие компании по мере того, как все больше программных продуктов переходит с десктопов в веб.

Патрик Коупленд в своем предисловии описал, как зарождалась методология Google. Она и сейчас гармонично развивается вместе с компанией. Для специалистов, которые приходят из других компаний, Google становится местом, где неэффективные методы переплавляются в полезные. Чем больше появлялось тестировщиков, тем больше новых идей и приемов мы пробовали. Балласт мы отбросили, а кое-что смогли перековать в прочный металл, ставший частью остова Google. Наши тестировщики готовы пробовать что угодно, но способны легко отказываться от неэффективных стратегий.

В основе Google лежат скорость работы и инновации. Мы выпускаем код именно в тот момент, когда он действительно востребован, а недовольных пользователей мало. Мы плотно взаимодействуем с первыми испытателями продукта, чтобы собирать как можно больше обратной связи. Тестирование в таких условиях тоже должно быть стремительным, а методы, требующие дотошного предварительного планирования или постоянного сопровождения, не выживут. В какой-то момент тестирование настолько тесно переплетается с разработкой, что эти две дисциплины невозможно отделить одну от другой. А иногда тестирование происходит настолько независимо, что разработчики даже ничего не подозревают.

НА ЗАМЕТКУ

В какой-то момент тестирование настолько тесно переплетается с разработкой, что эти две дисциплины невозможно отделить одну от другой. А иногда тестирование происходит настолько независимо, что разработчики даже ничего не подозревают.

С ростом Google стремительный темп работы замедлился совсем чуть-чуть. Нам потребовалось не больше года, чтобы создать операционную систему, клиентские приложения (типа Chrome) появляются ежемесячно, а уж веб-приложения меняются ежедневно. Хотя мы уже давно перестали быть стартапом, мы сохранили свойственные тому времени темпы. В таких условиях проще описать, каким тестирование не должно быть: догматичным, трудоемким, сложно выполнимым и занимающим много времени, — чем пытаться рассказать, каким оно должно быть. Хотя мы пытаемся нашей книгой сделать именно это. Одно можно сказать точно: тестирование не должно ставить палки в колеса инновациям и разработке. Второго шанса у него не будет.

Успех Google в тестировании нельзя списать на маленькое количество тестируемых продуктов. Размер и сложность тестирования Google не уступают задачам других компаний. Google выпускает почти все типы программ — от клиентских операционных систем до веб-, мобильных, корпоративных, коммерческих и социальных приложений. Наши приложения масштабны и сложны, ими пользуются сотни миллионов людей, мы постоянно «на мышке» хакеров. Значительная часть нашего исходного кода открыта. При этом у нас много устаревшего кода. Нас часто проверяют на соответствие требованиям законодательства. Наши продукты работают в сотнях стран и на многих языках. Пользователи ждут, что продукты Google окажутся простыми в использовании и будут «просто работать». То, чем тестировщики Google занимаются каждый день, нельзя назвать легкими задачами. Тестировщики Google сталкиваются со всеми сложными проблемами, какие только можно представить.

Мы готовы обсуждать, насколько идеально (на самом деле нет) организован процесс Google. Но в одном я точно уверен: наш подход к тестированию отличается от остальных. Вслед за стремительным переходом программ с десктопов в облака вполне реально, что подход «как в Google» будет набирать популярность в отрасли. Мои соавторы и я надеемся, что книга раскроет магическую формулу Google и положит начало обсуждению того, как отрасль должна решать задачу создания надежного ПО, на которое сможет положиться весь мир. Возможно, у методологии Google есть свои недостатки, но мы все равно хотим опубликовать ее и сделать доступной международному сообществу тестировщиков, чтобы она могла развиваться и совершенствоваться.

Подход Google на первый взгляд парадоксален: у нас во всей компании меньше выделенных тестировщиков, чем у многих наших конкурентов в одной команде разработки. Мы не делаем ставку на миллионную армию рядовых тестировщиков.

Мы — небольшой отряд элитного спецназа, успех которого зависит от превосходной тактики и современного вооружения. Как и в случае со спецназом, наш секретный ингредиент — ограниченность ресурсов. Мы следуем завету Ларри Пейджа, который сказал, что «дефицит приносит ясность», и это заставляет нас правильно расставлять приоритеты. Для всего, от описания функциональности до техник тестирования, у нас есть высокоэффективные приемы достижения поставленной цели — качества продукта. Дефицит заставляет ценить ресурсы тестирования и относиться к ним с уважением. Это помогает людям оставаться вовлеченными в развитие нашей отрасли. Когда меня спрашивают, в чем секрет нашего успеха, я всегда даю совет: «Не нанимайте слишком много тестировщиков».

НА ЗАМЕТКУ

Когда меня спрашивают, в чем секрет нашего успеха, я всегда даю совет: «Не нанимайте слишком много тестировщиков».

Как Google справляется с таким маленьким штатом тестировщиков? Если бы мне нужно было ответить просто, я бы сказал, что в Google вся ответственность за качество лежит на плечах тех, кто пишет код. Качество никогда не бывает проблемой «какого-то тестировщика». Каждый, кто пишет код в Google, — уже немногого тестировщик, а качество — это проблема всего коллектива (рис. 1.1). Говорить о соотношении численности «разработчики/тестировщики» в Google — то же самое, что рассуждать о чистоте воздуха на поверхности Солнца. Эту тему бессмысленно поднимать. Каждый инженер является и тестировщиком. Если в его должностях есть слово «тестирование», то он помогает другим специалистам проводить качественное тестирование.



Рис. 1.1. Программисты Google предпочитают качество широте функциональности

Мы выпускаем первоклассные программные продукты. Это свидетельствует о том, что наша формула достойна внимания. Возможно, какие-то ее части сработают и в других компаниях. Конечно, что-то может быть усовершенствовано. Ниже я привожу краткое описание нашей формулы. В последующих главах мы подробно проанализируем ее и разберемся, как же организовать процесс тестирования в царстве разработчиков.

Качество ≠ Тестирование

Фраза «тестирование не определяет качество» уже настолько избита, что просто обязана быть правдой. В любой области разработки, от автомобилей до софта, не получится отточить продукт до совершенства, если он изначально неправильно сконструирован. Спросите любого автопроизводителя, который хоть раз отзывал партию своих машин, чего стоят запоздальные попытки прикрутить качество на готовое изделие. Делайте все правильно с самого начала, не создавайте себе лишних трудностей.

Тем не менее это только звучит просто.

С одной стороны, качество и не создается тестированием, с другой — без тестирования сделать качественный продукт невозможно. Как вы узнаете, насколько качественный ваш продукт, не проводя тестирование?

Эта головоломка решается легко: перестаньте рассматривать разработку и тестирование по отдельности. Тестирование и разработка идут рука об руку. Напишите немного кода и протестируйте его. Затем напишите еще чуть-чуть и снова протестируйте. Тестирование не отдельная практика, это часть самого процесса разработки. Одним только тестированием качества не добиться. Рецепт получения высокого качества: смешивайте разработку и тестирование в блендере, пока они не станут единой субстанцией.

НА ЗАМЕТКУ

Одним только тестированием качества не добиться. Рецепт получения высокого качества: смешивайте разработку и тестирование в блендере, пока они не станут единой субстанцией.

Вот чего мы добивались: мы хотели объединить дисциплины разработки и тестирования, чтобы одной нельзя было заниматься без другой. Немножко программируем, потом тестируем. Еще немного пишем код и снова тестируем. Здесь важно, кто тестирует. Тестировщиков в классическом понимании у нас мало. Значит, кто должен тестировать? Разработчик. Кто лучше всего протестирует свой код? Тот, кто его написал. Кто больше всех заинтересован в коде без багов?

Правильно. Вот почему Google обходится таким небольшим числом тестировщиков — за качество отвечают разработчики. Если продукт сломается после выпуска, то шишки полетят в разработчика, создавшего проблему, а не в тестировщика, который ее не нашел.

В итоге качество достигается предотвращением, а не выявлением багов. Качество — часть разработки, а не тестирования. Интегрируя тестирование в разработку, мы создали поэтапный процесс, в котором можно легко откатить изменение, в котором слишком много багов. Так мы предотвращаем возникновение проблем у пользователей и сокращаем количество выделенных тестировщиков, которые выявляют баги, ставящие под угрозу выпуск продукта. В Google тестирование определяет, насколько хорошо мы предотвращаем ошибки.

Наш подход к созданию продуктов подразумевает слияние разработки и тестирования. Это подтверждают комментарии к коду типа «где твои тесты, чувак?», развесенные в туалетах плакаты с правилами тестирования¹ и многое другое. Тестирование должно стать неотделимым от разработки. Качество родится только тогда, когда разработка и тестирование начнут жить вместе.

НА ЗАМЕТКУ

Тестирование должно стать неотделимым от разработки. Качество родится только тогда, когда разработка и тестирование начнут жить вместе.

Роли

Чтобы девиз «Сам построил, сам и ломай» работал (и работал долго), вам нужна не только традиционная роль разработчика кода, но и другие роли. Конкретнее, должны появиться инженеры, которые помогут разработчикам тестировать эффективно и правильно. Они часто скромничают, называя себя *тестировщиками*, но на самом деле обеспечивать продуктивность — вот их основная задача. Тестировщики нужны для того, чтобы разработчики работали более продуктивно. Причем рост продуктивности основан на предотвращении появления ошибок из-за небрежной разработки. Так качество становится частью этой продуктивности. Все эти роли мы подробно рассмотрим в следующих главах, а пока обойдемся кратким описанием.

Роль **разработчика** (Software Engineer, SWE) всем знакома и привычна. Он пишет код функциональности приложений, который поставляется пользователям. Он создает проектную документацию, определяет структуры данных и общую

¹ <http://googletesting.blogspot.com/2007/01/introducing-testing-on-toilet.html>

архитектуру, а большую часть времени пишет код и проводит код-ревью¹. Разработчик пишет много тестового кода, например во время написания тестов для TDD и юнит-тестирования, и, как будет показано дальше в этой главе, участвует в создании малых, средних и больших тестов. Разработчик отвечает за качество всего кода, к которому он прикасается: пишет, исправляет или вносит изменения. Да, все верно: если разработчик должен изменить функцию и его изменение нарушает существующий тест или требует написания нового, он должен написать этот тест. Практически 100% рабочего времени разработчик пишет программный код.

Роль разработчика в тестировании (Software Engineer in Test, SET) тоже связана с разработкой, но фокусируется на тестируемости кода и создании инфраструктуры тестирования. Разработчик в тестировании анализирует архитектуру, уделяет особое внимание качеству кода и рискам проекта. Он выполняет рефакторинг, чтобы сделать код тестируемым, пишет фреймворки юнит-тестирования и средства автоматизации. Разработчик в тестировании работает с тем же кодом, что и разработчик, но больше заинтересован в улучшении качества и тестового покрытия, чем в добавлении новых фич или повышении производительности. Разработчик в тестировании тоже проводит почти 100% своего времени за написанием кода, но делает это ради повышения качества, а не для реализации фич для пользователей.

НА ЗАМЕТКУ

Разработчик в тестировании работает с тем же кодом, что и разработчик, но занимается скорее повышением качества и тестовым покрытием, чем добавлением новых фич или повышением производительности. Разработчик в тестировании пишет код, который помогает разработчику тестировать написанные им фичи.

Роль инженера по тестированию (Test Engineer, TE) связана с ролью разработчика в тестировании, но здесь на первом месте находятся пользователи и только на втором — разработчики. Некоторые инженеры по тестированию в Google пишут много кода для автотестов и управления сценариями использования и даже для имитации действий пользователя. Кроме того, именно они организуют работу по тестированию, которую выполняют другие инженеры, управляют выполнением тестов и интерпретируют их результаты тестов. Особенно важна их работа на поздних стадиях проекта, когда напряжение с приближением выпуска растет. Инженеры по тестированию — это эксперты продукта, консультанты по качеству и специалисты по анализу рисков. Одни пишут много кода, другие — мало².

1 Код-ревью (или рецензирование кода) — систематическая проверка исходного кода с целью обнаружения и исправления ошибок, допущенных при его написании. Эта практика позволяет находить ошибки раньше и способствует улучшению общего качества кода. — Примеч. перев.

2 Везде, где мы дальше пишем «тестировщики», мы имеем в виду именно роль инженера по тестированию. — Примеч. перев.

НА ЗАМЕТКУ

Инженеры по тестированию фокусируются на тестировании с точки зрения пользователя. Они занимаются общей организацией контроля качества, управляют выполнением тестов, интерпретируют их результаты и строят сквозную автоматизацию тестирования.

Итак, с точки зрения качества разработчики отвечают за фичи приложения и их качество отдельно от всего остального. Они ответственны за то, чтобы архитектура была устойчивой к ошибкам, приложение восстанавливалось после сбоев, отвечают за TDD, юнит-тесты и вместе с разработчиками в тестировании пишут код для тестирования фич.

Разработчики в тестировании отвечают за фичи тестирования. Они настраивают среду для изолированного выполнения кода с помощью имитации реального рабочего окружения. Для этого они создают такие компоненты, как заглушки (stubs), подставные объекты (mocks) и имитации (fakes), — мы рассмотрим их позже. Настроить очередь для управления отправкой кода в репозиторий — тоже их задача. Другими словами, разработчики в тестировании пишут код, который помогает разработчикам тестировать написанные ими фичи. Большую часть тестирования выполнит сам разработчик. Разработчики в тестировании в основном следят за тем, чтобы функциональность можно было легко протестировать, а разработчики не ленились писать тест-кейсы.

Получается, что разработчики в тестировании работают для разработчиков. Их конечная цель — качество кода и отдельных фич. Для этого они создают разработчикам удобную среду для тестирования кода. О пользователях думают тестировщики.

Если разработчики провели модульное и функциональное тестирование хорошо, то следующая задача — понять, насколько хорошо их исходный код вместе с данными будет работать на благо пользователей. Тестировщики проверяют добросовестность разработчиков дважды. Любые очевидные баги свидетельствуют о том, что тестирование раннего цикла было недостаточным или небрежным. Если таких багов мало, тестировщик переходит к своей основной задаче — убедиться в том, что продукт выполняет пользовательские сценарии, соответствует ожиданиям по производительности, что он надежен, правильно локализован и т. д. Тестировщики много тестируют сами и вдобавок обеспечивают координацию с другими инженерами, внешними тестировщиками-подрядчиками, тестировщиками из сообщества, внутренними пользователями, бета- и ранними внешними пользователями. Они сводят воедино все риски, которые могут сыграть из-за недочетов в базовой архитектуре, сложности функциональности и отказов в системе предотвращения сбоев. Как только тестировщики взялись за дело, их работе не видно конца.

Организационная структура

В большинстве компаний, где я работал, разработчики и тестировщики относились к одной команде разработки продукта. И разработчики, и тестировщики подчинялись одному и тому же руководителю проекта. Один продукт, одна команда — все участники говорят на одном языке.

К сожалению, я еще не видел, чтобы такой подход работал на практике. Обычно командой руководят менеджеры, вышедшие из среды программирования или управления, но не из среды тестировщиков. В предрелизном аврале они предпочтут выпустить функциональность полностью и закрыть как можно больше задач по «отделке» приложения, чем позаботиться об общем качестве. Внутри команды тестированию часто отводится второстепенная роль по отношению к разработке. Это отлично заметно по обилию глючных продуктов и преждевременных выпусков в нашей отрасли. Кто тут заказывал сервис-пак?

НА ЗАМЕТКУ

Совместными командами обычно руководят менеджеры, вышедшие из среды программирования или управления, но не из среды тестировщиков. В предрелизном аврале они предпочтут выпустить функциональность полностью и закрыть как можно больше задач по «отделке», чем позаботиться об общем качестве. Внутри команды тестированию часто отводится второстепенная роль по отношению к разработке.

Давайте разберемся, кто кому подчиняется в Google. У нас есть своя иерархия, которая делится на направления, называемые Focus Area (FA). У нас есть направления Client (проекты Chrome, Google Toolbar и т. д.), Geo (проекты Maps, Google Earth и т. д.), Ads, Apps, Mobile и т. д. Все разработчики подчиняются директору или вице-президенту направления.

Правда, тестирование ломает эту схему. Тестирование находится в отдельном горизонтальном направлении, которое отвечает за продуктивность разработки. Направление продуктивности разработки существует параллельно с продуктовыми направлениями. Тестировщиков, по сути, предоставляют командам разработки продуктов во временное пользование. Они свободно поднимают вопросы безопасности и освещают области, где нет тестов или много багов. Так как мы не подчиняемся команде разработки продукта, нам нельзя ответить, что для нас есть дела поважнее. Мы сами устанавливаем свои приоритеты, и они всегда в рамках обеспечения надежности, безопасности и всего такого, пока мы не решим, что продукту нужно что-то другое. Если команда разработки захочет провести тестирование по упрощенной схеме, это необходимо согласовать заранее, а мы всегда можем сказать «нет».

С такой структурой работы мы можем сохранять небольшой штат тестировщиков. Команда разработки продукта не может снизить техническую планку тести-

ровщиков или нанять их больше, чтобы свалить на них рутинную работу. Рутинная работа вокруг фичи входит в обязанности разработчика, ее нельзя переложить на какого-нибудь тестировщика-несчастливчика.

Тестировщиков на проекты назначают руководители направления продуктивности разработки. Они принимают стратегические решения, опираясь на приоритеты компании, сложность продукта и потребности конкретной команды проекта в сравнении с другими. Конечно, они могут ошибаться, и такое случается. Но в целом такая схема обеспечивает баланс ресурсов между фактическими, а не предполагаемыми потребностями.

НА ЗАМЕТКУ

Тестировщиков на проект назначают руководители направления продуктивности разработки. Они принимают стратегические решения, опираясь на приоритеты компании, сложность продукта и потребности конкретной команды проекта в сравнении с другими. Такая схема сохраняет баланс ресурсов между фактическими, а не предполагаемыми потребностями. Централизованность подходов, идей и решений помогает всей компании использовать самые удачные из них.

Статус временно выделяемых на проект тестировщиков упрощает их перемещение по проектам. Они не только сохраняют свежий взгляд и остаются в курсе дел, но и быстро распространяют удачные идеи по компании. Прием или инструмент тестирования, который хорошо сработал в продукте направления Geo, скорее всего, пойдет в работу снова, когда этот тестировщик перейдет на Chrome. Нет более быстрого способа распространения инноваций, чем перемещение самих новаторов.

Мы считаем, что тестировщику достаточно 18 месяцев работы над одним продуктом, а потом он может (если, конечно, хочет) перейти в другую команду. Конечно, есть и ложка дегтя — команда теряет накопленный опыт. Но это компенсируется появлением тестировщиков с широким кругозором, знакомых с разными продуктами и технологиями. В Google много тестировщиков, понимающих клиентские, браузерные и мобильные технологии, способных эффективно программировать на многих языках и для разных платформ. А поскольку все продукты и услуги Google сейчас интегрированы друг с другом теснее, чем когда-либо, тестировщики легко перемещаются в пределах компании и их опыт применим везде, независимо от проекта.

Ползти, идти, бежать

Одна из ключевых причин, почему Google успешен даже с маленьким количеством тестировщиков, в том, что мы редко пытаемся сразу поставить большой набор фич. Как раз наоборот. Мы строим минимально жизнеспособный продукт и выпускаем

его в тот момент, когда он может стать полезным как можно большему количеству людей, получаем обратную связь и дальше развиваем продукт итеративно. Так мы действовали с Gmail, у которого ярлык «бета-версии» сохранялся четыре года. Мы предупреждали пользователей, что продукт все еще дорабатывается. Ярлык «бета-версии» мы убрали, только когда достигли поставленной цели в 99,99% нормальной работоспособности почты. По той же схеме мы работали с Android при выпуске G1 — продукта, получившего хорошие отзывы. Он был улучшен и обогащен новыми фичами, когда выходил для линейки телефонов Nexus. Важно помнить, что ранняя версия, за которую платят пользователи, должна быть достаточно функциональной и полезной для них. То, что версия является ранней, еще не означает, что она должна быть слабым продуктом.

НА ЗАМЕТКУ

Google часто выпускает «минимально полезный продукт» как исходную версию и собирает обратную связь от внутренних и внешних пользователей. А потом быстро выдает обновления, тщательно анализируя качество на каждом этапе. Прежде чем попасть к пользователям, продукты проходят через разные каналы: канареечный, разработки, тестирования, бета- и выпуска.

Это не такое уж жульничество, как может показаться на первый взгляд. Чтобы дойти до состояния канала бета-версии, продукту надо пройти через ряд других каналов и доказать свою ценность. Для Chrome — продукта, над которым я работал первые два года в Google, — мы использовали разные каналы в зависимости от нашей уверенности в качестве продукта и от объема отзывов, который нам был нужен. Последовательность примерно такая.

- **Канареечный канал:** используется для ежедневных сборок, которые, на наш взгляд, еще не готовы для выпуска. Помните канареек, с которыми спускались в шахты для обнаружения токсичного газа? Если ежедневная сборка оказывается нежизнеспособной (канарейка умирает, если есть хоть небольшая примесь газа в воздухе), это свидетельствует о том, что процесс разболтался и работу надо пересмотреть. Канареечные версии подходят только для самых преданных пользователей, которые экспериментируют с продуктом. Они не для тех, кто использует приложение для реальной работы. Как правило, сборки из канареечного канала используют только инженеры (разработчики и тестировщики) и менеджеры, работающие над продуктом.

НА ЗАМЕТКУ

Команда Android пошла еще дальше — телефоны основной команды разработки почти постоянно работают на ежедневной сборке. Идея в том, что разработчики будут писать меньше кода с багами, если от этого зависит их возможность позвонить домой.

- **Канал разработки:** используется разработчиками в повседневной работе. Обычно это еженедельные сборки, которые стабильно использовались какое-то время и прошли набор тестов (об этом подробнее в следующих главах). Все инженеры, работающие над продуктом, *обязаны* взять сборку канала разработки и использовать ее в реальной работе, чтобы проверить при продолжительном применении. Если сборка из канала разработки не-пригодна для нормальной работы, она отправляется обратно в канареечный канал. Это неприятная ситуация, которая потребует от инженеров серьезного пересмотра своей работы.
- **Тестовый канал:** сюда попадает лучшая сборка месяца — та, которая прошла наиболее продолжительное тестирование и которой доверяют инженеры. Сборку тестового канала можно выкатывать для внутренних пользователей. Это кандидат на сборку бета-канала, если покажет хорошие результаты при долгосрочном использовании. Однажды сборка тестового канала станет достаточно стабильной для использования внутри компании, а иногда мы даже передаем ее внешним сотрудникам и партнерам, которым может быть полезно раннее знакомство с продуктом.
- **Бета-канал или канал выпуска:** сюда попадают только стабильные сборки, успешно прошедшее внутреннее использование и удовлетворяющие всем критериям качества, установленным командой. Это первые сборки, которые доступны пользователям.

Метод «ползти, идти, бежать» помогает нам проводить тесты и экспериментировать с приложением на ранней стадии разработки. В дополнение к данным ежедневных автоматических тестов мы получаем ценную обратную связь от настоящих пользователей.

Виды тестов

Вместо того чтобы разделять тестирование на модульное, интеграционное и системное, мы делим все тесты на *малые, средние и большие*. Пожалуйста, не путайте с методом оценки из гибких методологий. Мы ориентируемся на охват, а не на размер теста. Малые тесты покрывают малые объемы кода, средние — объемы побольше и т. д. Любой инженер, независимо от своей роли, может заниматься любыми типами тестов. Их можно проводить в ручном или запускать в автоматическом режиме. Реальный опыт говорит, что чем меньше тест, тем скорее он будет автоматизирован.

НА ЗАМЕТКУ

Вместо того чтобы разделять тестирование на модульное, интеграционное и системное, мы делим все тесты на малые, средние и большие, исходя из их охвата, а не размера.

Малые тесты чаще всего (хотя и не всегда) автоматизируются. Они исполняют код одной функции или модуля. Обычно они проверяют типичные функциональные проблемы, повреждение данных, неверные условия и ошибки, связанные со сдвигом значений на единицу. Малые тесты выполняются быстро, за несколько секунд и меньше. Как правило, их пишут разработчики, реже — разработчики в тестировании и почти никогда — инженеры по тестированию. Для выполнения малых тестов обычно нужна среда с подставными объектами и имитациями. Подставные объекты и имитации относятся к заглушкам (то есть заменителям реальных функций), они работают как заменители объектов в зависимостях — несуществующих, слишком ненадежных или затрудняющих эмуляцию ошибочных ситуаций. Мы расскажем об этом подробнее в следующих главах. Инженеры по тестированию редко пишут малые тесты, но могут выполнять их, диагностируя конкретные сбои. Малые тесты отвечают на вопрос «*Делает ли этот код то, что он должен делать?*»

Средние тесты обычно автоматизируются. Они покрывают две или больше функции. Тесты фокусируются на том, чтобы проверить взаимодействие между функциями, которые вызывают друг друга или контактируют напрямую. Такие функции мы называем *ближайшими соседями*. Разработчик в тестировании управляет разработкой средних тестов на ранней стадии цикла продукта. Их пишут, исправляют и сопровождают разработчики. Если тест не проходит, разработчик чинит его сам. На более поздней стадии разработки инженеры по тестированию могут выполнять средние тесты вручную (если это сложно или дорого автоматизировать) или автоматически. Средние тесты отвечают на вопрос «*Взаимодействуют ли соседние функции друг с другом так, как должны?*»

Большие тесты покрывают не меньше трех (а обычно больше) функций. Это реальные пользовательские сценарии, которые используют реальные источники данных, а их выполнение может занять несколько часов и даже больше. Они затрагивают и интеграцию в целом, но чаще всего большие тесты проверяют, насколько приложение соответствует потребностям пользователей. Все три роли вовлечены в создание больших тестов. Их можно проводить разными способами, от автоматизированного до исследовательского ручного тестирования. Большие тесты отвечают на вопрос «*Работает ли продукт так, как нужно пользователю, и дает ли желаемый результат?*» Сквозные сценарии, которые выполняются на завершенном продукте, относятся к большим тестам.

НА ЗАМЕТКУ

Малые тесты проверяют один программный блок в полностью имитированной среде. Средние тесты проверяют взаимодействие модулей в имитированной или реальной среде. Большие тесты проверяют любое количество модулей в реальной среде с настоящими, не имитированными ресурсами.

Не так критично, как именно вы называете тесты, главное, чтобы все понимали эти термины одинаково¹. Здесь важно то, что тестировщики Google говорят на одном языке, описывая, что именно тестируется и в каком объеме. Когда кто-то особо изобретательный говорит о четвертой разновидности тестов, которую они прозвали *громадными тестами*, любой инженер в компании сразу представит себе общесистемный тест, который выполняется очень долго и проверяет все функциональные аспекты. Ничего больше пояснить не требуется².

Мы определяем, в каком объеме проводить тестирование и что именно тестиировать, по-разному для каждого конкретного продукта. Google предпочитает выпускать часто и стремится как можно раньше передать продукт пользователям, чтобы быстрее получить обратную связь и внести необходимые изменения. Google тратит много сил на то, чтобы создавать продукты, которыми люди действительно пользуются. Мы выкатываем новые фичи как можно раньше, чтобы они начали приносить пользу. Плюс мы избегаем лишних затрат на фичи, которые не нужны пользователям, потому что вовремя об этом узнаем. Для этого мы подключаем пользователи и внешних разработчиков в процесс как можно раньше. Так мы понимаем, соответствует ли наш продукт их ожиданиям.

Наконец, выбирая между автоматизированным и ручным тестированием, мы отдаляем предпочтение первому. Если это можно автоматизировать и проблема не требует человеческого внимания и интуиции, то это нужно автоматизировать. Только проблемы, которые явно требуют оценки человеком (например, красив ли пользовательский интерфейс или не нарушает ли раскрытие данных конфиденциальность), должны доставаться ручному тестированию.

НА ЗАМЕТКУ

Выбирая между автоматизированным и ручным тестированием, мы отдааем предпочтение первому. Если это можно автоматизировать и проблема не требует человеческого внимания и интуиции, то это нужно автоматизировать.

-
- 1 Классификация на малые, средние и большие тесты была выбрана для стандартизации. Многие тестировщики пришли к нам из разных компаний, в которых понятия «смоук-тестирования», BVT (Build Verification Test), интеграционных тестов и т. д. имели разные, часто противоположные значения. Старые термины приносили с собой столько хаоса, что я решил ввести новую терминологию.
 - 2 Кстати, концепция громадных тестов формализована, и инфраструктура автоматизации Google использует понятия малых, средних тестов (и далее по списку) для определения последовательности автоматизированного выполнения. Эта тема более подробно рассматривается в главе, посвященной разработчикам в тестировании.

Замечу, что в Google выполняется очень много ручного тестирования, как сценарного, так и исследовательского, но даже оно проходит под зорким присмотром автоматизации. Технологии записи преобразуют ручные тесты в автоматизированные, которые снова прогоняются на последующих сборках, чтобы отловить регрессионные баги и дать тестировщикам возможность переключиться на новые проблемы. Мы автоматизируем отправку баг-репортов и распределение задач ручного тестирования¹. Например, если автоматизированный тест не проходит, система определяет последнее изменение кода, которое считает наиболее вероятной причиной, отправляет сообщение его авторам и автоматически заводит баг. Максимально приблизить автоматизацию к человеческим возможностям — это техническое задание для следующего поколения инструментов тестирования, создаваемых в Google.

¹ Технологии записи Google и ручное тестирование со вспомогательной автоматизацией подробно описаны в главах, посвященных роли инженеров по тестированию.

Разработчик в тестировании

Давайте представим идеальный процесс разработки. Все начинается с теста. Вот код, который построил разработчик Джек. А вот — тест, который разработчик Джек придумал еще до того, как написал код. Другими словами, до того, как написать первую строчку кода, разработчик прикидывает, что ему понадобится для тестирования. Затем он напишет тесты для граничных значений, для слишком больших и слишком малых входных данных, для значений, нарушающих граничные условия, и для множества других предположений. Какие-то из этих тестов станут частью функций, превратятся в самотестируемый код или юнит-тесты. На этом уровне лучше всех тестирует код тот, кто его написал. Иначе говоря, в коде, который построил Джек, хорошо разбирается сам разработчик Джек, и он же протестирует его лучше всех.

Другие тесты требуют знаний, выходящих за рамки кода, и зависят от внешней инфраструктуры. Например, у нас есть тест, который возвращает данные из удаленного хранилища (с сервера баз данных или из облака). Для тестирования нам нужна либо сама база данных, либо ее имитация. В индустрии уже выработались инструменты для этого: *тестовая оснастка* (*harness*), *тестовая инфраструктура*, *подставные объекты* (*mocks*) и *имитации* (*fakes*). В мире идеального процесса разработки эти макеты сразу существуют для любого интерфейса, с которым имеет дело разработчик, и каждый аспект любой функции можно протестировать в любое время. Но не увлекайтесь, мы находимся в воображаемом мире.

Мы подошли к первой развилке, где нашему сказочному миру разработки потребовался еще один герой, то есть тестировщик. При написании кода функциональности и кода тестов важны разные образы мышления. Это разные

типы разработки. При разработке функционального кода на первом плане стоит *создание*. Нужно принимать во внимание пользователей, их сценарии использования продукта, последовательности действий и т. д. А при написании тестового кода есть ориентир на *разрушение*, то есть разработку такого кода, который выявит случаи, мешающие эффективной работе пользователя и его действиям. Так как мы находимся в сказочной стране идеальной разработки, мы можем нанять двух Джеков, один из которых построит дом, а другой покажет, как его можно сломать.

НА ЗАМЕТКУ

При написании кода функциональности и кода тестов важны разные образы мышления.

В нашем сказочном мире идеального процесса разработки у нас будет сколько угодно разработчиков функциональности и разработчиков тестов, которые идут рука об руку и вместе строят сложный программный продукт. Это еще присказка, ведь настоящая сказка позволит нам выделить по целому разработчику на каждую фичу, к каждому разработчику приставить столько разработчиков тестов, сколько нужно. Они занимались бы глобальной тестовой инфраструктурой, помогали бы решить проблемы, найденные юнит-тестированием, чтобы разработчики не отвлекались на них от процесса создания, требующего полной концентрации.

Итак, одни разработчики пишут функциональный код, другие разработчики пишут тестовый код, и тут мы вспоминаем про третью сторону: сторону пользователя.

Естественно, в нашем сказочном мире эта задача упадет на плечи отдельных инженеров. Их задачи связаны с пользователем: сценарии использования продукта, пользовательские истории, исследовательское тестирование и т. д. Разработчики со стороны пользователя рассматривают то, как разные фичи связываются воедино и образуют единое целое. Они работают над проблемами всей системы и обычно принимают точку зрения пользователя, проверяя, действительно ли совокупность частей образует что-то полезное для них.

Итак, вот наше представление об идеальной разработке ПО: три разные роли разработчиков работают вместе над надежным, практичным совершенством, причем каждый специализируется на чем-то своем и все взаимодействуют на равных.

Кто хочет работать в компании, в которой программные продукты создаются подобным образом? Мы точно хотим!

К сожалению, никто из нас в таких компаниях не работает. Google, как и многие компании до нее, потратил множество усилий на то, чтобы приблизиться к идеалу. Возможно, потому что мы поздно начали, мы учились на ошибках предшественников. Google повезло поймать момент перехода модели программных продуктов от огромных клиентских приложений с многолетними циклами выпуска к облачным сервисам, которые выпускаются каждые несколько недель, дней или

часов¹. Благодаря удачному стечению обстоятельств у нас получилось хоть как-то приблизить разработку в Google к идеальному процессу разработки ПО.

Итак, разработчики в Google занимаются реализацией функциональности, отвечают за построение компонентов, строят основу приложения, поставляемого пользователю. Они пишут функции и код юнит-тестов для этих функций. *Джек строит дом.*

Разработчики в тестировании в Google отвечают за создание средств тестирования. Они помогают разработчикам с написанием юнит-тестов и создают большие тестовые инфраструктуры, упрощая разработчикам процесс создания малых и средних тестов и помогая выявлять больше проблем. *Джек делает дом устойчивым.*

Инженеры по тестированию в Google встают на сторону пользователя во всех аспектах, связанных с качеством. В контексте разработки они создают автоматизацию пользовательских сценариев, а в контексте продукта — оценивают универсальность и эффективность всех действий по тестированию, которые выполняют другие инженеры. *Джек готовит дом к приходу гостей.*

Это уже не сказка, а наша попытка сделать ее былью.

В этой книге мы рассказываем в основном о работе разработчиков в тестировании и инженеров по тестированию. Роль разработчиков мы затрагиваем только там, где она соприкасается с тестированием. На самом деле разработчики вовлечены в тестирование довольно сильно, но обычно эти процессы курируют специалисты, в должности которых есть слово «тестирование».

Жизнь разработчика в тестировании

Когда компания только появляется, тестировщиков в ней, как правило, нет. Точно так же как нет руководителей проектов, системных администраторов и других должностей. Каждый сотрудник выполняет все эти роли одновременно. Мы любим представлять, как Ларри и Сергей² ломали головы над пользовательскими сценариями и юнит-тестами на заре Google. С ростом компании появились разработчики в тестировании, которые первыми привнесли фокус на качество в строго технологический дух программирования³.

1 Заметим, что даже для клиентских продуктов Google старается делать частые и надежные обновления с помощью автообновления, встроенного во все клиентские приложения.

2 Речь идет о Ларри Пейдже и Сергее Брине — основателях Google.

3 О том, как появилась роль разработчика в тестировании, Патрик Коупленд рассказывает в предисловии.

Как организованы процессы разработки и тестирования

Прежде чем мы взглянем поближе на задачи разработчиков в тестировании, давайте посмотрим, в каких условиях они работают. Разработчики в тестировании и разработчики связаны очень тесно, у них много общих задач в работе над любым продуктом. Так происходит, потому что в Google тестирование — обязанность всей инженерной команды, а не только людей, в должностях которых есть слово «тестирование».

Готовый код — основной артефакт, над которым работают все участники команды. Организовать структуру, написать код и сопровождать его — их ежедневные задачи. Большая часть кода Google хранится в едином репозитории и использует общие инструменты. Все процессы сборки и выпуска построены вокруг этих инструментов и репозитория. Каждый инженер Google знает эту среду как свои пять пальцев, поэтому любой участник команды независимо от своей роли может залить новый код, создать и запустить тест, собрать версию и т. д.

НА ЗАМЕТКУ

Готовый код — основной артефакт, над которым работают все участники команды. Организовать структуру, написать код и сопровождать его — их ежедневные задачи.

Еще один плюс единого репозитория: инженерам, переходящим из проекта в проект, не нужно переучиваться, а так называемые «двадцатипроцентные сотрудники» могут нормально работать с первого дня в проекте¹. Кроме того, весь исходный код доступен любому инженеру. Разработчики веб-приложений могут просмотреть интересующий их код браузера, не спрашивая ни у кого разрешения, или узнать, как другие, более опытные сотрудники решали аналогичные задачи. Можно взять код для повторного использования на уровне модулей или даже на уровне структур контроллов или данных. Google — един, и он использует общий репозиторий с удобными (еще бы!) средствами поиска.

1 В Google есть официальная система распределения графика, благодаря которой любой сотрудник может тратить 20% своего рабочего времени на любой другой проект Google. Иными словами, четыре дня в неделю вы занимаетесь своей основной работой, а один день экспериментируете и изобретаете. Неизбежно именно так распределять свое время, многие бывшие сотрудники Google вообще считали такой рабочий график мифическим. Однако каждый из авторов этой книги участвовал в «двадцатипроцентных проектах», значит, они — реальность. Более того, многие инструменты, описанные в книге, — результат «двадцатипроцентной» работы, которая со временем воплотилась в полноценные финансируемые проекты. Правда, многие сотрудники Google в свое «двадцатипроцентное время» просто работают с другими проектами, а не занимаются экспериментами, поэтому от такой концепции рабочего времени выигрывают многие. (К моменту издания книги на русском языке Google официально отменил эту систему. — Примеч. перев.)

Благодаря тому что наша кодовая база открыта, доступ к ней есть у всей компании, а технический инструментарий един, мы разработали огромный набор библиотек и сервисов для общего использования. Общий код надежно работает на боевой среде Google и ускоряет работу над проектами, а использование общих библиотек при разработке сокращает количество багов.

НА ЗАМЕТКУ

Благодаря тому что наша кодовая база открыта, доступ к ней есть у всей компании, а технический инструментарий един, мы разработали огромный набор библиотек и сервисов для общего использования.

Так как код вливается в общую инфраструктуру, к его разработке инженеры относятся очень осторожно. Подтверждают такое особое отношение неписаные, но соблюдаемые правила работы с кодом.

- Используйте уже написанные библиотеки по максимуму. Пишите свою только в том случае, если у вас есть на то серьезная причина из-за специфических особенностей проекта.
- Когда вы пишете код, помните, что он должен быть легко читаем. Сделайте так, чтобы его можно было без труда найти в репозитории. С вашим кодом будут работать другие люди, поэтому нужно сделать его понятным и легко изменяемым.
- Общий код должен быть автономным и пригодным для повторного использования. Инженеров поощряют за создание сервисов, которые могут использовать другие команды. Возможность повторного использования кода гораздо важнее, чем его сложность.
- Сделайте зависимости настолько очевидными, что их невозможно будет пропустить. Если проект зависит от общего кода, то пусть о любых его изменениях узнают участники всех зависимых проектов.

Если инженер предлагает более удачное решение, то он же проводит рефакторинг всех существующих библиотек и помогает перевести все зависимые проекты на новые библиотеки. Работа на благо сообщества должна поощряться¹.

Google серьезно относится к процедуре код-ревью, и любой код, особенно общий, просматривает специалист с черным поясом по читаемости кода. Специальная комиссия выделяет таких специалистов за умение писать чистый и сти-

¹ Самым распространенным механизмом такого поощрения в Google является премия «от коллег». Любой инженер, которому помогла работа другого инженера, может назначить ему премию в благодарность. У руководителей тоже есть свои способы поощрения сотрудника. Смысл в том, что работу на благо сообщества нужно подпитывать! Конечно, не стоит забывать, что есть и неформальный способ поблагодарить коллегу – «услуга за услугу».

листически точный код для четырех основных языков Google (C++, Java, Python и JavaScript).

Код в общем репозитории устанавливает более высокую планку для тестирования, — об этом мы расскажем чуть позже.

Для решения проблем с зависимостями платформ мы минимизируем их различия на машинах сотрудников. У всех инженеров на компьютерах стоит та же версия OS, что и на боевых машинах. Мы тщательно следим за актуальностью сборок Linux, чтобы разработчик, проводящий тестирование на своем компьютере, получил такие же результаты, как и при тестировании в боевой среде. Различия в процессорах и операционных системах между компьютерами инженеров и дата-центрами минимальны¹. Если баг возник на компьютере тестировщика, то он, скорее всего, воспроизведется и на компьютере разработчика, и в условиях реальной эксплуатации.

Весь код, связанный с зависимостями платформ, собирается в библиотеки на самом нижнем уровне стека. Управляют этими библиотеками те же ребята, которые отвечают за дистрибутивы Linux. Наконец, для каждого из языков программирования, на которых пишут в Google, мы используем только один компилятор, который поддерживается и постоянно тестируется на одной из сборок Linux. Ничего сложного, но эта простая схема экономит силы на финальных стадиях тестирования, а заодно сокращает количество проблем, связанных со спецификой среды, которые сложны в отладке и отвлекают от разработки новых функций. Простота и надежность.

НА ЗАМЕТКУ

Вся платформа Google построена на простоте и единобразии: одинаковые сборки Linux для компьютеров инженеров и машин с боевой средой, общие базовые библиотеки под централизованным управлением, общая инфраструктура хранения исходного кода, сборки и тестирования, один компилятор для каждого из языков программирования, общая спецификация сборки для всех языков. И самое важное, культура, которая уважает и поощряет поддержку этих общих ресурсов.

Тему единобразия платформ и единства репозитория продолжает единая система сборки, не зависящая от языка, на котором написан проект. Не важно, на каком языке работает команда (C++, Python или Java), она все равно будет использовать общие «файлы сборки».

Чтобы сборка состоялась, нужно указать «цель сборки». Это может быть библиотека, бинарный файл или набор тестов, который состоит из некоторого количества исходных файлов.

¹ Это правило в Google нарушено только в лабораториях локального тестирования Android и Chrome OS, где для проверки новых сборок нужно иметь широкий спектр оборудования.

Последовательность шагов следующая.

1. Напишите класс или набор функций в одном или нескольких исходных файлах. Убедитесь, что весь код компилируется.
2. Укажите цель сборки (например, определенную библиотеку) для новой сборки.
3. Напишите юнит-тесты, которые импортируют библиотеку, имитируют нетривиальные зависимости и выполняют интересующие нас пути в коде для самых актуальных входных данных.
4. Создайте тестовую сборку для юнит-тестов.
5. Соберите и запустите сборку с тестами. Изменяйте код до тех пор, пока все тесты не будут проходить.
6. Запустите все обязательные инструменты статического анализа, которые проверяют соответствие кода гайдлайнам и выявляют стандартные баги.
7. Отправьте итоговый код на код-ревью (подробнее о код-ревью мы расскажем позже), внесите изменения и повторите все юнит-тесты.

В результате мы создаем две сборки: собственно библиотеку, представляющую новый сервис, и сборку тестов для этого сервиса. Учтите, что многие разработчики в Google применяют методологию TDD (Test-Driven Development, или разработка через тестирование), при которой шаг 3 предшествует шагам 1 и 2.

Если разработчик конструирует более крупный сервис, то он продолжает писать код, связывая постоянно увеличивающиеся сборки библиотек. Сборка бинарника создается из основного файла, который ссылается на нужные библиотеки. Так появляется продукт Google, у которого есть:

- хорошо протестированный автономный бинарный файл;
- легкочитаемая и приспособленная для повторного использования библиотека (с набором вспомогательных библиотек, которые можно использовать для создания других сервисов);
- набор юнит-тестов, покрывающих нужные аспекты всех сборок.

Типичный продукт Google — это набор нескольких сервисов. В любой команде мы стараемся добиться соотношения 1:1 между разработчиками и сервисами. Это означает, что сервисы собираются и тестируются параллельно, а затем интегрируются в итоговой сборке. Чтобы связанные сервисы могли создаваться одновременно, их интерфейсы взаимодействия согласовываются в начале проекта. Тогда разработчики могут реализовывать зависимости через такие интерфейсы, а не через библиотеки. В начале работы разработчики создают имитации таких интерфейсов, чтобы начать писать тесты на уровне всего сервиса.

Разработчики в тестировании вовлечены в создание большинства тестовых сборок и определяют, где нужно писать малые тесты. По мере того как маленькие сборки собираются в целое приложение, задачи растут, и уже нужно проводить более крупные интеграционные тесты. Если для сборки одной библиотеки достаточно выполнения малых тестов, написанных самим разработчиком, то с увеличением объема сборок разработчики в тестировании вовлекаются во все большей степени и пишут уже средние и большие тесты.

Сборка увеличивается в размерах, и малые тесты становятся частью регрессионного пакета. Они должны быть всегда актуальны. В противном случае в них инициируются баги, отладка которых ничем не отличается от отладки багов основного кода. Тесты — часть функциональности, а значит, баги в тестах относятся к функциональным багам и исправляются. Такая схема гарантирует, что новая функциональность не нарушит работу существующей, а изменения в коде не сломают тесты.

Разработчики в тестировании — центр всей этой деятельности. Они помогают разработчикам решить, какие юнит-тесты написать. Они создают подставные объекты и имитации. Они пишут средние и большие интеграционные тесты. Именно об этих задачах разработчиков в тестировании мы собираемся сейчас рассказать.

Кто такие разработчики в тестировании на самом деле?

Разработчики в тестировании — это инженеры, которые помогают тестировать на всех уровнях процесса разработки Google. Но все же в первую очередь они именно разработчики. Во всех наших руководствах по найму и внутренних документах написано, что их работа на 100% связана с программированием. Этот специфический, можно даже сказать гибридный, подход к тестированию позволяет нам рано привлекать тестировщиков к проектам. Причем они занимаются не составлением абстрактных тест-планов или моделей качества, а сразу погружаются в проектирование и написание кода. Это ставит на одну чашу весов и программистов, и тестировщиков. Это повышает производительность команды и создает доверие ко всем видам тестирования, включая ручное и исследовательское, которое потом проведут уже другие инженеры.

НА ЗАМЕТКУ

Тест — это еще одна фича приложения, и за нее отвечают разработчики в тестировании.

Разработчики в тестировании работают рука об руку с разработчиками продукта, причем в буквальном смысле. Мы стараемся, чтобы они даже сидели вместе. Тесты — это еще одна фича приложения, за которую отвечают разработчики в тестировании. Разработчики и разработчики в тестировании участвуют в реview кода, написанного друг другом.

На собеседовании разработчики в тестировании должны продемонстрировать такие же знания по программированию, как и разработчики. Даже больше — они должны уметь тестировать код, который написали. Проще говоря, разработчик в тестировании должен ответить на те же вопросы по программированию, что и разработчик, а потом еще решить задачки по тестированию.

Как вы уже догадались, специалистов на эту роль найти непросто. Скорее всего, это и есть причина относительно малого количества разработчиков в тестировании в Google. А вовсе не то, что мы нашли волшебную формулу производительности. Скорее, мы смирились с реальностью и адаптировали нашу работу, зная, что такое сочетание навыков встречается редко. Однако сходство ролей разработчика и разработчика в тестировании дало приятный побочный эффект: люди могут переходить из одной группы в другую. Google как раз старается поддерживать переходы между ролями. Представьте компанию, в которой все разработчики умеют тестировать, а все тестировщики умеют программировать. Нам далеко до этого, и, наверное, мы никогда такими не станем, но эти группы все-таки пересекаются. Мы находим разработчиков в тестировании со склонностью к разработке и разработчиков со склонностью к тестированию. Такие ребята становятся нашими лучшими инженерами и образуют самые эффективные команды разработки.

Ранняя стадия проекта

В Google нет правила, когда именно разработчики в тестировании должны присоединиться к проекту. Так же как нигде не прописано, когда именно проект становится «реальным». Типичный сценарий создания нового проекта такой: эксперимент, над которым работали в «двадцатипроцентное» время, набирается сил и становится самостоятельным продуктом Google. Именно так развивались Gmail и Chrome OS. Эти проекты начинались с неформальных идей, а со временем выросли в полноценные продукты со своими командами разработчиков и тестировщиков. Наш друг Альберто Савоя (написавший введение к этой книге) любит повторять, что «качество не имеет значения, пока ваш продукт не имеет значения».

В свое «двадцатипроцентное» время команды придумывают много нового. Часть этих идей ни к чему не приведет, другая часть станет новыми фичами других проектов, а некоторые смогут перерасти в официальные продукты Google. Ни одному проекту по умолчанию не полагается тестирование. Проект может потерпеть

неудачу, поэтому включать в него тестировщиков — напрасная трата ресурсов. Если проект закроют, что мы будем делать с готовой тестовой инфраструктурой?

Браться за качество еще до того, как концепция продукта дозрела и полностью сформирована, — это типичный пример неправильной расстановки приоритетов. Мы повидали много прототипов, созданных в «двадцатипроцентное» время, которые перерабатывались так сильно, что на стадии бета-версии или версии для внутренних пользователей от оригинального кода оставалась крохотная часть. Тестирование в экспериментальных проектах — безнадежная затея.

Конечно, не стоит впадать в крайности. Если продукт слишком долго развивается без тестирования, то становится тяжело менять архитектурные решения, плохо влияющие на его тестируемость. Это усложняет автоматизацию, а тестовые инструменты становятся ненадежными. Чтобы повысить качество, придется многое переделывать. Такой технический долг может затормозить разработку продукта на годы.

В Google не принято, чтобы тестирование появлялось в проектах рано. На самом деле разработчики в тестировании часто приходят на проекты на ранних этапах, но пока они больше разработчики, чем тестировщики. Это наше сознательное решение, но это не значит, что на ранних стадиях мы забываем о качестве. Это следствие неформального и одержимого новшествами процесса созидания в Google. Много-месячное планирование проекта перед разработкой, включающее контроль качества и тесты, — это не про нас. Проекты в Google начинаются намного менее формально.

Chrome OS — яркий пример такого подхода. На этом проекте все трое авторов этой книги работали больше года. Но задолго до того, как мы официально присоединились к работе, несколько разработчиков создали прототип. Он состоял в основном из скриптов и заглушек, но зато позволял продемонстрировать идею «чисто браузерного» приложения руководству Google, чтобы официально утвердить проект. На стадии прототипа команда концентрировалась на экспериментах и хотела доказать, что этот концепт вообще жизнеспособен. Тратить время на тестирование — или даже проектировать с оглядкой на тестируемость — было бы неразумным, особенно учитывая, что проект был еще неофициальным, а все демосценарии в будущем все равно заменили бы реальным кодом. Когда сценарии выполнили свое предназначение и продукт был утвержден, руководитель разработки обратился к нам за помощью в тестировании.

Все это — особая культура Google. Ни один проект не получит ресурсы тестирования просто так. Команды разработки обращаются к тестировщикам за помощью, убеждая их в том, что проект по-настоящему интересен и перспективен. После того как руководители разработки Chrome OS обрисовали свой проект, состояние дел и график выпуска, мы смогли выдвинуть свои требования по участию разработчиков в тестировании, уровню покрытия кода юнит-тестами и разделению обязанностей в процессе работы. Мы не участвовали в зарождении проекта, но когда он стал реальным, мы смогли серьезно повлиять на его реализацию.

НА ЗАМЕТКУ

Ни один проект не получит ресурсы тестирования просто так. Команды разработки обращаются к тестировщикам за помощью, убеждая их в том, что проект по-настоящему интересен и перспективен.

Структура команды

Разработчики часто глубоко погружены в код, который пишут, и сосредоточены на одной фиче продукта или даже ее части. Они принимают все решения исходя из локальной пользы, воспринимая продукт очень узко. Хороший разработчик в тестировании должен делать ровно наоборот: смотреть на продукт широко и держать в голове общую картину продукта, со всеми его фичами. Более того, он должен понимать, что разработчики приходят, делают свою работу и уходят, а продукт должен продолжать жить дальше.

Проектам типа Gmail или Chrome суждено пройти через много версий. Над ними будут трудиться сотни разработчиков. Представим, что разработчик присоединился к команде на третьей версии продукта. Если этот продукт хорошо задокументирован и пригоден к тестированию, если у него есть стабильный работоспособный механизм автоматизации тестов, если есть процессы, по которым легко добавить новый код, — считайте, что те ранние разработчики в тестировании сработали хорошо.

Со всем этим постоянным наращиванием функциональности, выпуском новых версий и патчей, переименованием и переработкой бывает трудно понять, когда работа над продуктом завершается. Но совершенно точно у каждого продукта есть четкая отправная точка. Здесь, в начале, мы формируем свои цели, планируем и пробуем. Мы даже пытаемся документировать то, что, как мы думаем, мы будем делать. Мы стремимся принимать такие решения, которые будут жизнеспособны в долгосрочной перспективе.

Чем больше экспериментов, прикодов и набросков планов мы сделали до начала реализации проекта, тем сильнее наша уверенность в долгой и успешной жизни проекта. Но надо знать меру. С одной стороны, мы не хотим планировать настолько мало, что потом это нам аукнется. С другой стороны, не хочется потратить несколько недель только на то, чтобы в конце понять, что условия изменились или оказались совсем не такими, какими их представляли. Поэтому на ранней фазе разумно вести документацию и структурировать процессы, но объем этой работы определяют сами инженеры из команды проекта.

Сначала в команде разработки нового продукта появляется ведущий инженер и еще несколько других технических ролей. В Google неформальное звание «ведущий инженер» получает специалист, который отвечает за выбор технического

направления работ, координирует проект и становится его главным техническим представителем для других команд. Он знает ответы на все вопросы о проекте или может перенаправить их правильному человеку. Ведущим инженером продукта обычно становится разработчик или любой другой инженер, который выступает в роли разработчика.

Ведущий инженер со своей командой начинает с набросков первого проектного документа (об этом мы расскажем в следующем разделе). Постепенно документ растет, а это значит, что пора привлекать инженеров разных специализаций. Многие команды просят разработчика в тестировании еще на старте, несмотря на то что их мало.

Проектная документация

У каждого проекта в Google есть основной проектный документ. Это живой документ, он развивается вместе с проектом. Вначале этот документ описывает цель проекта, предпосылки его создания, предполагаемый список участников и архитектурных решений. На раннем этапе участники команды вместе дополняют этот документ. В больших проектах может понадобиться создать более мелкие проектные документы для основных подсистем. К концу этой стадии набор проектных документов должен стать основой для плана всех будущих работ. Документ могут помочь проанализировать ведущие инженеры других команд из предметной области проекта. Когда все дополнения и замечания собраны, фаза проектирования завершается и официально начинается стадия реализации.

Очень хорошо, если разработчики в тестировании присоединяются к проекту на раннем этапе. Тогда они сделают важную работу, которая заметно повлияет на результаты. Если они правильно разыграют карты, то упростят жизнь участников проекта, ускорив ход работы. У разработчиков в тестировании есть очень важное преимущество в команде: по сравнению с другими инженерами *они обладают самым широким представлением о продукте*. Хороший разработчик в тестировании добавляет свою ценность к работе узкоспециализированных разработчиков и влияет на весь проект в целом, а не просто пишет код. Не разработчики, а именно разработчики в тестировании обычно выявляют общие схемы для повторного использования кода и взаимодействия компонентов. Оставшаяся часть этого раздела как раз и рассказывает о том, какую ценную работу могут выполнять разработчики в тестировании на ранней стадии проекта.

НА ЗАМЕТКУ

Цель разработчика в тестировании на ранней стадии — упростить жизнь других участников проекта, ускорив выполнение работы.

Ничто не заменит свежего взгляда со стороны в нашей работе. Перед тем как отправить проектный документ на официальную оценку, разработчикам очень полезно получить отзывы от своих коллег. Хороший разработчик в тестировании всегда готов помочь с этим и даже специально выделяет для этого рабочее время. Если нужно, он добавляет разделы, посвященные качеству и надежности. Вот почему такой подход оправдан.

- Разработчик в тестировании должен знать структуру тестируемой системы, и чтение проектной документации в этом помогает. Так что рецензирование полезно как разработчику в тестировании, так и разработчику.
- Чем раньше внесено предложение, тем больше вероятность, что оно попадет в документ, а потом и в код. Так разработчика в тестировании сможет сильнее повлиять на проект.
- Разработчик в тестировании станет первым человеком, рецензирующим все проектные документы, и не пропустит ни одной итерации. Его знание проекта в целом сможет сравниться только со знанием ведущего инженера.
- Это прекрасный шанс установить рабочие отношения со всеми инженерами, с чьим кодом разработчик в тестировании будет работать, когда начнется разработка.

Проектную документацию нужно рецензировать целенаправленно и вдумчиво, а не бегло просматривать, как утреннюю газету. Хороший разработчик в тестировании, оценивая документ, преследует четкие цели. Вот что советуем мы:

- **Полнота.** Выделяйте части документа, где не хватает информации или нужны особые знания, которые не особо распространены в команде, особенно это важно, если в команде есть новички. Попросите автора документа закончить раздел или добавить ссылку на дополнительную документацию.
- **Грамотность.** Не пропускайте грамматические, орфографические и пунктуационные ошибки. Небрежность плохо скажется на будущем коде. Не создавайте почву для небрежности.
- **Согласованность.** Убедитесь в том, что текст соответствует диаграммам. Проследите, чтобы документ не противоречил утверждениям, сделанным в других документах.
- **Архитектура.** Проанализируйте архитектуру, предложенную в документе. Можно ли ее реализовать с доступными ресурсами? Какая инфраструктура будет использоваться? Прочтайте описание этой инфраструктуры и изучите ее подводные камни. Можем ли мы поддерживать предложенную инфраструктуру в нашей системе? Архитектура не слишком сложная? Можно ее

упростить? Не слишком ли она проста? Что еще нужно учесть при работе с этой архитектурой?

- **Интерфейсы и протоколы.** Четко ли в документе определены будущие протоколы? Полнотью ли описаны интерфейсы и протоколы, которые будет предоставлять продукт? Соответствуют ли эти интерфейсы и протоколы своим целям? Соответствуют ли они стандартам продуктов Google? Можно ли рекомендовать разработчику пойти дальше и начать писать protobuf-файлы (этую концепцию мы опишем дальше)?
- **Тестирование.** Насколько тестопригодна система, описанная в документе? Нужно ли будет встраивать в код новые зацепки для тестирования? Если да, проследите за тем, чтобы это добавили в документацию. Можно ли скорректировать структуру системы для упрощения тестирования или использования готовой тестовой инфраструктуры? Оцените, что нужно сделать для тестирования системы, и договоритесь с разработчиками, чтобы эту информацию добавили.

НА ЗАМЕТКУ

Рецензировать проектные документы нужно вдумчиво и целенаправленно, а не бегло. Рецензирование преследует четкие цели.

Когда разработчик в тестировании обсуждает результаты рецензирования с автором документации — разработчиком, они оценивают объем работы по тестированию и обсуждают, как распределить эту работу между ролями. Это подходящий момент, чтобы задокументировать, как и зачем разработчикам писать юнит-тесты и какие приемы будет использовать команда, чтобы хорошенько протестировать продукт. Если такая дискуссия конструктивна, значит работа началась успешно.

Интерфейсы и протоколы

С описанием интерфейсов и протоколов разработчики Google справляются легко, ведь для этого нужно писать их любимый код. В Google разработали специальный расширяемый язык Protocol Buffer¹ для сериализации структурированных данных. Protobuf — это механизм описания данных, который не зависит от языка программирования или платформы, которые вы собираетесь использовать. По сути, он похож на XML, только компактнее, быстрее и проще. Разработчик определяет

¹ Protocol Buffer Google имеет открытую спецификацию, которая представлена тут: <http://code.google.com/apis/protocolbuffers/>

структурой данных с помощью Protocol Buffer и потом использует генерированные исходники для чтения и записи структурированных данных на разных языках (Java, C++ или Python). Часто код Protocol Buffer становится первым написанным кодом в проекте. Можно встретить документацию, которая ссылается на protobuf-файлы при описании того, как должна работать полностью реализованная система.

Разработчик в тестировании тщательно анализирует protobuf-код, потому что вскоре ему придется реализовывать большинство интерфейсов и протоколов, описанных в этом коде. Все верно, именно разработчик в тестировании обычно делает эту работу. Необходимость в интеграционном тестировании часто возникает до того, как будут построены все зависимые подсистемы, и к этому надо быть готовым. Чтобы проводить интеграционное тестирование настолько рано, разработчик в тестировании создает имитации и заглушки нужных зависимостей каждого компонента. Интеграционные тесты все равно придется написать, и чем раньше они будут написаны, тем больше от них пользы. Подставные объекты и имитации пригодятся для интеграционного тестирования и дальше. Через них гораздо проще имитировать условия возникновения ошибки и сам сбой, чем через боевую систему.

НА ЗАМЕТКУ

Чтобы проводить интеграционное тестирование как можно раньше, разработчик в тестировании создает имитации и заглушки нужных зависимостей каждого компонента.

Планирование автоматизации

Время разработчика в тестировании ограничено и расписано по минутам, поэтому хорошая идея — создавать план автоматизации как можно раньше. План должен быть реалистичным. Пытаться автоматизировать все сразу в одном тестовом пакете — это ошибка. У разработчиков такие наполеоновские планы обычно не вызывают восторга, и они не спешат помогать. Если разработчик в тестировании хочет заручиться поддержкой разработчика, план автоматизации должен быть простым, четким и способным повлиять на проект. Тяжело поддерживать масштабную автоматизацию, которая с ростом системы расшатывается еще больше. Разработчиков можно привлечь писать только узконаправленные автотесты, которые приносят пользу с самого начала.

Не стоит слишком рано вкладываться в сквозную автоматизацию — она привязывает вас к конкретной архитектуре проекта и не имеет смысла, пока продукт не сформировался и не стал стабильным. Если вы начали слишком рано и собрали много информации, она все равно обесценится к концу проекта, потому что уже поздно будет менять архитектуру продукта. Время, которое разработчик в testi-

ровании мог бы уделить шлифовке качества, было потрачено на сопровождение неустойчивых сквозных тестов.

НА ЗАМЕТКУ

Не стоит слишком рано вкладываться в сквозную автоматизацию — она привязывает вас к конкретной архитектуре проекта.

В Google разработчики в тестировании подходят к планированию так. Сначала мы выделяем интерфейсы, которые, как нам кажется, могут содержать баги. Мы создаем подставные объекты и имитации, чтобы контролировать взаимодействие с этими интерфейсами и обеспечить хорошее тестовое покрытие.

На следующем шаге мы строим легковесный фреймворк автоматизации, который даст нам возможность запустить систему подставных объектов. При таком подходе любой разработчик, код которого использует один из наших подставных интерфейсов, может создать себе отдельную сборку и прогонять на ней автоматизированные тесты перед тем, как заливать изменения в репозиторий. Только хорошо протестированный код попадает в репозиторий. Это одно из ключевых достоинств автоматизации: плохой код не попадает в экосистему и не загрязняет общую кодовую базу.

План автоматизации должен не только перечислить средства автоматизации, которые создает разработчик в тестировании: подставные объекты, имитации и фреймворки. План должен объяснять, как все участники проекта будут получать информацию о качестве сборки. Мы включаем в план создание механизмов отчетности и панели мониторинга результатов тестов и статуса выполнения. Наши разработчики в тестировании увеличивают шансы создания высококачественного кода, упрощая процесс его разработки и делая его более прозрачным.

Тестируемость

Разработчики в тестировании плотно работают вместе с разработчиками. Пока разработчики пишут код функциональности и тесты для него, разработчики в тестировании создают для них тестовые фреймворки, а заодно выполняют часть работы по ее сопровождению. Ответственность за качество делится между этими ролями поровну.

Основная цель разработчиков в тестировании — сделать продукт *тестируемым*. Они дают рекомендации, как выстроить структуру программы и стиль написания кода, чтобы упростить будущее юнит-тестирование. Они создают удобную среду тестирования, чтобы разработчики могли тестировать сами. Об этом чуть позже, а сейчас поговорим о том, как пишется код в Google.

Чтобы прийти к равноправной ответственности разработчиков и разработчиков в тестировании за исходный код, мы в Google строим процесс разработки вокруг код-ревью. О том, как рецензировать код, говорят даже больше, чем о том, как его писать.

Рецензирование кода — полноценный этап работы разработчиков. У него есть свои инструменты и своя культура, которая строится на концепции коммитеров, как в опенсорс-сообществах, где коммитить код в базу могут только самые надежные и доказавшие это правда разработчики.

НА ЗАМЕТКУ

Google строит процесс разработки вокруг код-ревью. О том, как рецензировать код, говорят даже больше, чем о том, как его писать.

В Google любой инженер может стать *коммитером*. Мы пользуемся концепцией *читаемости* кода, чтобы отличать проверенных сотрудников от новичков. Вот как работает этот процесс.

Когда код написан, он упаковывается в пакет, который мы называем *списком изменений*. Дальше он отправляется для рецензирования в приложение, которое в Google называют Mondrian, в честь голландского художника, положившего начало абстрактному искусству. Mondrian отсылает код ответственному разработчику или разработчику в тестировании для окончательного утверждения¹.

Блоки нового кода, изменения в существующем коде, исправления багов — все это может входить в список изменений. Размеры списков могут варьироваться от пары строк кода до нескольких сотен, причем большие списки почти всегда разбиваются на несколько мелких, чтобы рецензентам было удобнее.

Новички рано или поздно получают от коллег бейдж «спец по легкочитаемому коду», если постоянно коммитят качественные списки изменений. Эти бейджи — разные для разных языков программирования. Основные языки в Google — C++, Java, Python и JavaScript. Бейдж указывает нам опытного разработчика, который старается писать так, чтобы вся кодовая база однородной, будто ее писал один разработчик².

Прежде чем список изменений попадет к рецензенту, он пройдет ряд автоматических проверок в Mondrian. Программа проверит выполнение простых условий, например насколько код соответствует гайдлайнам стиля программирования Google, и более сложных, например что все тесты, связанные с этим списком изменений, проходят. Тесты для списка почти всегда включены прямо в него — тестовый и функциональный код живут вместе. Выполнив проверку, Mondrian

1 Версия Mondrian с открытым кодом на базе App Engine доступна по адресу: <http://code.google.com/p/rietveld/>

2 Руководство Google по стилю для C++ доступно по адресу: <http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml>

отправит рецензенту по электронной почте ссылку на списки изменений. В свою очередь рецензент проанализирует код и выдаст рекомендации его автору. Процесс повторяется до тех пор, пока все рецензенты не будут довольны и автоматическая проверка не будет проходить гладко.

Дальше код встает в очередь на отправку, цель которой — поддерживать сборку в состоянии «зеленый свет», в котором все тесты проходят. Это последняя линия защиты между системой непрерывной сборки проекта и системой контроля версий. Код собирается и тестируется на чистой среде, поэтому здесь отлавливаются баги, которые на машинах разработчиков могли не обнаружиться. Эти конфигурационные баги могли бы нарушить процесс непрерывной сборки или, хуже того, пробраться в систему контроля версий.

Очередь на отправку позволяет участникам больших команд совместно работать в ветке `main` дерева исходного кода. Больше не нужно замораживать код на время интеграции веток и прохождения тестов. Получается, что разработчики больших команд могут работать так же эффективно и независимо, как если бы команда была маленькая и гибкая. Только у разработчика в тестировании прибавляется работы — ведь скорость написания и заливки кода в репозиторий увеличивается.

Как появились очереди на отправку и непрерывная сборка

Джефф Карролло

Когда-то Google был маленьким. Тогда казалось, что провести юнит-тестирование перед коммитом изменений вполне достаточно. Но даже тогда случалось, что тесты не проходили, и люди тратили свое время на поиск и решение проблем.

Компания росла. Чтобы масштабироваться, наши разработчики писали и поддерживали качественные библиотеки и инфраструктуру, которые использовали все команды. Со временем росло количество, размеры и сложность базовых библиотек. Код проектов стал интенсивно использовать сторонние библиотеки и инфраструктуру, и одних юнит-тестов стало недостаточно — уже требовалось интеграционное тестирование. В какой-то момент стало понятно, что многие баги вызывались зависимостями между компонентами. Так как тесты не запускались до тех пор, пока кому-нибудь не вздумывалось закоммитить изменение в своем проекте, интеграционные баги могли оставаться незамеченными по несколько дней.

Потом мы пришли к панели мониторинга юнит-тестов. Система автоматически считала каждый каталог верхнего уровня в дереве кода компании

«проектом». Плюс каждый мог определить свой «проект», в котором связывал сборки кода с тестами и назначал ответственного за сопровождение. Каждый день система прогоняла все тесты по всем проектам. Система записывала статистику прохождений каждого теста и показывала ее на главной панели. Если тесты падали, ответственные за их сопровождение получали письма каждый день, поэтому тесты оставались неисправными недолго. Тем не менее проблемы оставались.

Ежедневного прогона всех тестов оказалось недостаточно — команды хотели быстрее отлавливать разрушительные изменения. Некоторые команды начали писать скрипты непрерывной сборки, которые непрерывно делали сборку и выполняли юнит- и интеграционные тесты на отдельных машинах. Осознав, что эту систему можно сделать общей для всех команд, Крис Лопес и Джей Корбетт сели и написали «Систему непрерывной сборки Криса и Джея». Теперь любой проект мог развернуть свою систему непрерывной сборки. Достаточно было просто зарегистрировать машину, заполнить файл конфигурации и запустить скрипт.

Система быстро стала популярной, и вскоре большинство проектов в Google перешло на нее. Если тест не проходил, то программа оповещала всех ответственных за изменение по почте. О сбоях стали узнавать через несколько минут после коммита изменений в базу кода. Кроме того, система отмечала «Золотые списки изменений» — контрольные точки в системе контроля версий, в которых успешно проходили все тесты проекта. Теперь разработчики могли ориентироваться на стабильную версию исходников без недавних проблемных изменений. Это очень помогало при выборе стабильной сборки для выпуска.

Но и этого инструмента инженерам оказалось недостаточно. Команды становились больше, проекты — сложнее, потери от поломанных сборок росли. Разработчики строили новые очереди отправок, чтобы защитить системы непрерывной сборки. В ранних реализациях все списки изменений действительно вставали в очередь: система тестировала и одобряла или отклоняла списки последовательно. Если нужно было провести много продолжительных тестов подряд, то между постановкой списка изменений в очередь и его фактической передачей в систему контроля версий могло пройти несколько часов. В следующих версиях уже реализовали параллельное выполнение ожидающих списков изменений, но они запускались изолированно друг от друга. Хотя это могло создавать проблемы нарушения последовательности потоков, такие случаи были редки, их оперативно обнаруживала система непрерывной сборки. Возможность заливки кода через несколько минут после отправки запроса экономила много времени. Это компенсировало затраты на исправление редких падений системы непрерывной сборки.

Так большинство крупных проектов Google перешло на использование очередей на отправку. Во многих командах даже выделяли специального человека на роль «смотрителя сборки», задача которого заключалась в том, чтобы быстро реагировать на любые проблемы, выявленные очередью проверки или системой непрерывной сборки.

Эти две системы, панель мониторинга юнит-тестов и система непрерывной сборки Криса и Джая, использовались в Google несколько лет. Они принесли огромную пользу командам, были несложны в настройке и не-прихотливы в сопровождении. И вот встал вопрос о реализации этих систем в виде общей инфраструктуры для всех команд. Так появилась система Test Automation Program (ТАР). Когда мы писали эту книгу, ТАР уже заменила собой обе первоначальные системы. Ее используют почти все проекты Google, кроме Chromium и Android. Только проекты с открытым кодом используют отдельные деревья исходного кода и серверные среды сборки.

Плюсы того, что большинство сотрудников используют один набор инструментов и единую инфраструктуру, трудно переоценить. Одной простой командой инженер может собрать и исполнить все бинарники и тесты, которые связаны с его списком изменений, получить данные о покрытии кода, сохранить и проанализировать результаты в облаке, а потом посмотреть их в виде отчета на постоянной веб-странице. Результат выводится в терминал в виде сообщения «PASS» или «FAIL» со ссылками на подробную информацию. Когда разработчик выполняет тесты, их результаты и данные о покрытии кода сохраняются в облаке, и любой рецензент может посмотреть их через внутренний инструмент для код-ревью.

Пример работы разработчика в тестировании

Следующий пример объединяет все, о чем мы говорили выше. Предупреждаем, в этом разделе много технической информации с уймой низкоуровневых деталей. Если вам интересна только общая картина, смело переходите к следующему разделу.

Представьте простое веб-приложение, с помощью которого пользователи отправляют URL-адреса в Google для добавления в Google-индекс. Форма HTML содержит два поля — URL-адрес и комментарий — и генерирует запрос HTTP GET к серверу Google в следующем формате:

```
GET /addurl?url=http://www.foo.com&comment=Foo+comment HTTP/1.1
```

На стороне сервера это веб-приложение делится на две части: AddUrlFrontend, который получает запрос HTTP, распознает и проверяет его, и бэкенд AddUrlService.

Сервис бэкенда получает запросы от AddUrlFrontend, проверяет, нет ли в них ошибок, и дальше взаимодействует с такими хранилищами данных, как, например, Google Bigtable¹ или Google File System².

Разработчик начинает работу с создания каталога для проекта:

```
$ mkdir depot/addurl/
```

Затем он определяет протокол AddUrlService с использованием языка Protocol Buffers³:

FILE: DEPOT/ADDURL/ADDURL.PROTO

```
message AddUrlRequest {  
    required string url = 1;           // The URL address entered by user.  
    optional string comment = 2;       // Comments made by user.  
}  
  
message AddUrlReply {  
    // Error code if an error occurred.  
    optional int32 error_code = 1;  
    // Error message if an error occurred.  
    optional string error_details = 2;  
}  
  
service AddUrlService {  
    // Accepts a URL for submission to the index.  
    rpc AddUrl(AddUrlRequest) returns (AddUrlReply) {  
        option deadline = 10.0;  
    }  
}
```

В файле *addurl.proto* определены три важных элемента: сообщения AddUrlRequest и AddUrlReply и сервис удаленного вызова процедур (RPC, Remote Procedure) AddUrlService.

Посмотрев на определения сообщения AddUrlRequest, мы видим, что поле *url* должно быть задано вызывающей стороной, а поле *comment* не является обязательным.

Точно так же из определения сообщения AddUrlReply следует, что оба поля — *error_code* и *error_details* опционально могут быть переданы в ответах сервиса. Мы предполагаем, что в типичном случае, когда URL-адрес успешно принят, эти поля останутся пустыми, чтобы минимизировать объем передаваемых данных. Это одно из правил Google: типичный случай должен работать быстро.

1 <http://labs.google.com/papers/bigtable.html>

2 <http://labs.google.com/papers/gfs.html>

3 <http://code.google.com/apis/protocolbuffers/docs/overview.html>

Из определения `AddUrlService` видно, что сервис содержит единственный метод `AddUrl`, который принимает `AddUrlRequest` и возвращает `AddUrlReply`. По умолчанию вызов метода `AddUrl` прерывается по тайм-ауту через 10 секунд, если клиент не получил ответа за это время. Реализации интерфейса `AddUrlService` могут включать в себя сколько угодно систем хранения данных, но для клиентов интерфейса это несущественно, поэтому эти подробности не отражены в файле `addurl.proto`.

Обозначение '`= 1`' в полях сообщений не имеет никакого отношения к значениям этих полей. Оно существует для того, чтобы протокол можно было дорабатывать. Например, кто-то захочет добавить поле `uri` в сообщение `AddUrlRequest` к уже имеющимся полям. Для этого вносится следующее изменение:

```
message AddUrlRequest {  
    required string url = 1;      // The URL entered by the user.  
    optional string comment = 2;   // Comments made by the user.  
    optional string uri = 3;      // The URI entered by the user.  
}
```

Но это выглядит довольно глупо — скорее всего, потребуется просто переименовать поле `url` в `uri`. Если это число и тип останутся неизменными, сохранится совместимость между старой и новой версией:

```
message AddUrlRequest {  
    required string uri = 1;      // The URI entered by user.  
    optional string comment = 2;   // Comments made by the user.  
}
```

Написав файл `addurl.proto`, разработчик переходит к созданию правила сборки `proto_library`, которое генерирует исходные файлы C++, определяющие сущности из `addurl.proto`, и компилирует их в статическую библиотеку `addurl` C++. С дополнительными параметрами можно генерировать исходный код для языков Java и Python.

FILE: DEPOT/ADDURL/BUILD

```
proto_library(name="addurl",  
              srcs=["addurl.proto"])
```

Разработчик запускает систему сборки и исправляет все проблемы, обнаруженные ею в `addurl.proto` и в файле `BUILD`. Система сборки вызывает компилятор Protocol Buffers, генерирует исходные файлы `addurl.pb.h` и `addurl.pb.cc` и статическую библиотеку `addurl`, которую теперь можно подключить.

Пора писать `AddUrlFrontend`. Для этого мы объявляем класс `AddUrlFrontend` в новом файле `addurl_frontend.h`. Этот код в основном шаблонный.

FILE: DEPOT/ADDURL/ADDURL_FRONTEND.H

```
#ifndef ADDURL_ADDURL_FRONTEND_H_
#define ADDURL_ADDURL_FRONTEND_H_

// Forward-declaration of dependencies.
class AddUrlService;
class HTTPRequest;
class HTTPReply;

// Frontend for the AddUrl system.
// Accepts HTTP requests from web clients,
// and forwards well-formed requests to the backend.
class AddUrlFrontend {
public:
    // Constructor which enables injection of an
    // AddUrlService dependency.
    explicit AddUrlFrontend(AddUrlService* add_url_service);
    ~AddUrlFrontend();

    // Method invoked by our HTTP server when a request arrives
    // for the /addurl resource.
    void HandleAddUrlFrontendRequest(const HTTPRequest* http_request,
                                    HTTPReply* http_reply);

private:
    AddUrlService* add_url_service_;

    // Declare copy constructor and operator= private to prohibit
    // unintentional copying of instances of this class.
    AddUrlFrontend(const AddUrlFrontend&);
    AddUrlFrontend& operator=(const AddUrlFrontend& rhs);
};

#endif // ADDURL_ADDURL_FRONTEND_H_
```

Продолжая определять классы AddUrlFrontend, разработчик создает файл *addurl_frontend.cc*, в котором описывает класс AddUrlFrontend. Для экономии места мы опустили часть файла.

FILE: DEPOT/ADDURL/ADDURL_FRONTEND.CC

```
#include "addurl/addurl_frontend.h"
#include "addurl/addurl.pb.h"
#include "path/to/httpqueryparams.h"

// Functions used by HandleAddUrlFrontendRequest() below, but
// whose definitions are omitted for brevity.

void ExtractHttpQueryParams(const HTTPRequest* http_request,
                           HTTPQueryParams* query_params);
```

```

void WriteHttp200Reply(HTTPReply* reply);

void WriteHttpReplyWithErrorDetails(
    HTTPReply* http_reply, const AddUrlReply& add_url_reply);

// AddUrlFrontend constructor that injects the AddUrlService
// dependency.

AddUrlFrontend::AddUrlFrontend(AddUrlService* add_url_service)
    : add_url_service_(add_url_service) {
}

// AddUrlFrontend destructor - there's nothing to do here.
AddUrlFrontend::~AddUrlFrontend() {
}

// HandleAddUrlFrontendRequest:
// Handles requests to /addurl by parsing the request,
// dispatching a backend request to an AddUrlService backend,
// and transforming the backend reply into an appropriate
// HTTP reply.
//
// Args:
// http_request - The raw HTTP request received by the server.
// http_reply - The raw HTTP reply to send in response.

void AddUrlFrontend::HandleAddUrlFrontendRequest(
    const HTTPRequest* http_request, HTTPReply* http_reply) {

    // Extract the query parameters from the raw HTTP request.

    HTTPQueryParams query_params;
    ExtractHttpqueryParams(http_request, &query_params);

    // Get the 'url' and 'comment' query components.
    // Default each to an empty string if they were not present
    // in http_request.

    string url =
        query_params.GetQueryComponentDefault("url", "");
    string comment =
        query_params.GetQueryComponentDefault("comment", "");

    // Prepare the request to the AddUrlService backend.

    AddUrlRequest add_url_request;
    AddUrlReply add_url_reply;
    add_url_request.set_url(url);
}

```

```

if (!comment.empty()) {
    add_url_request.set_comment(comment);
}

// Issue the request to the AddUrlService backend.

RPC rpc;
add_url_service_->AddUrl(
    &rpc, &add_url_request, &add_url_reply);

// Block until the reply is received from the
// AddUrlService backend.

rpc.Wait();

// Handle errors, if any:

if (add_url_reply.has_error_code()) {
    WriteHttpReplyWithErrorDetails(http_reply, add_url_reply);
} else {

    // No errors. Send HTTP 200 OK response to client.

    WriteHttp200Reply(http_reply);
}
}

```

На функцию `HandleAddUrlFrontendRequest` ложится большая нагрузка — так устроены многие веб-обработчики. Разработчик может разгрузить эту функцию, выделив часть ее функциональности вспомогательным функциям. Но такой рефакторинг обычно не проводят, пока сборка не станет стабильной, а написанные юнит-тесты не будут успешно проходить.

В этом месте разработчик изменяет существующую спецификацию сборки проекта `addurl`, включая в нее запись для библиотеки `addurl_frontend`. При сборке создается статическая библиотека C++ для `AddUrlFrontend`.

FILE: /DEPOT/ADDURL/BUILD

```

# From before:
proto_library(name="addurl",
              srcs=["addurl.proto"])

# New:
cc_library(name="addurl_frontend",
            srcs=["addurl_frontend.cc"],
            deps=[
                "path/to/httpqueryparams",

```

```

    "other_http_server_stuff",
    ":addurl", # Link against the addurl library above.
])

```

Разработчик снова запускает средства сборки, исправляет ошибки компиляции и компоновщика в *addurl_frontend.h* и *addurl_frontend.cc*, пока все не будет собираться и компоноваться без предупреждений или ошибок. На этой стадии пора писать юнит-тесты для *AddUrlFrontend*. Они пишутся в новом файле *addurl_frontend_test.cc*. Тест определяет имитацию для бэкенд-системы *AddUrlService* и использует конструктор *AddUrlFrontend* для внедрения этой имитации во время тестирования. При таком подходе разработчик может внедрять ожидания и ошибки в поток операций *AddUrlFrontend* без изменения кода *AddUrlFrontend*.

FILE: DEPOT/ADDURL/ADDURL_FRONTEND_TEST.CC

```

#include "addurl/addurl.pb.h"
#include "addurl/addurl_frontend.h"

// See http://code.google.com/p/googletest/
#include "path/to/googletest.h"

// Defines a fake AddUrlService, which will be injected by
// the AddUrlFrontendTest test fixture into AddUrlFrontend
// instances under test.

class FakeAddUrlService : public AddUrlService {
public:
    FakeAddUrlService()
        : has_request_expectations_(false),
          error_code_(0) {
    }

    // Allows tests to set expectations on requests.
    void set_expected_url(const string& url) {
        expected_url_ = url;
        has_request_expectations_ = true;
    }

    void set_expected_comment(const string& comment) {
        expected_comment_ = comment;
        has_request_expectations_ = true;
    }

    // Allows for injection of errors by tests.

    void set_error_code(int error_code) {
        error_code_ = error_code;
    }
}

```

```

void set_error_details(const string& error_details) {
    error_details_ = error_details;
}

// Overrides of the AddUrlService::AddUrl method generated from
// service definition in addurl.proto by the Protocol Buffer
// compiler.

virtual void AddUrl(RPC* rpc,
                      const AddUrlRequest* request,
                      AddUrlReply* reply) {

    // Enforce expectations on request (if present).

    if (has_request_expectations_) {
        EXPECT_EQ(expected_url_, request->url());
        EXPECT_EQ(expected_comment_, request->comment());
    }

    // Inject errors specified in the set_* methods above if present.

    if (error_code_ != 0 || !error_details_.empty()) {
        reply->set_error_code(error_code_);
        reply->set_error_details(error_details_);
    }
}

private:

// Expected request information.
// Clients set using set_expected_* methods.

string expected_url_;
string expected_comment_;
bool has_request_expectations_;

// Injected error information.
// Clients set using set_* methods above.

int error_code_;
string error_details_;
};

// The test fixture for AddUrlFrontend. It is code shared by the
// TEST_F test definitions below. For every test using this
// fixture, the fixture will create a FakeAddUrlService, an
// AddUrlFrontend, and inject the FakeAddUrlService into that
// AddUrlFrontend. Tests will have access to both of these
// objects at runtime.

```

```

class AddurlFrontendTest : public ::testing::Test {

    // Runs before every test method is executed.

    virtual void SetUp() {

        // Create a FakeAddUrlService for injection.

        fake_add_url_service_.reset(new FakeAddUrlService);

        // Create an AddUrlFrontend and inject our FakeAddUrlService
        // into it.

        add_url_frontend_.reset(
            new AddUrlFrontend(fake_add_url_service_.get()));
    }
    scoped_ptr<FakeAddUrlService> fake_add_url_service_;
    scoped_ptr<AddUrlFrontend> add_url_frontend_;
};

// Test that AddurlFrontendTest::SetUp works.

TEST_F(AddurlFrontendTest, FixtureTest) {

    // AddurlFrontendTest::SetUp was invoked by this point.

}

// Test that AddUrlFrontend parses URLs correctly from its
// query parameters.

TEST_F(AddurlFrontendTest, ParsesUrlCorrectly) {
    HTTPRequest http_request;
    HTTPReply http_reply;

    // Configure the request to go to the /addurl resource and
    // to contain a 'url' query parameter.

    http_request.set_text(
        "GET /addurl?url=http://www.foo.com HTTP/1.1\r\n\r\n");

    // Tell the FakeAddUrlService to expect to receive a URL
    // of 'http://www.foo.com'.

    fake_add_url_service_->set_expected_url("http://www.foo.com");

    // Send the request to AddUrlFrontend, which should dispatch
    // a request to the FakeAddUrlService.
}

```

```

add_url_frontend_->HandleAddUrlFrontendRequest(
    &http_request, &http_reply);

// Validate the response.

EXPECT_STREQ("200 OK", http_reply.text());
}

// Test that AddUrlFrontend parses comments correctly from its
// query parameters.

TEST_F(AddurlFrontendTest, ParsesCommentCorrectly) {
    HTTPRequest http_request;
    HTTPReply http_reply;

    // Configure the request to go to the /addurl resource and
    // to contain a 'url' query parameter and to also contain
    // a 'comment' query parameter that contains the
    // url-encoded query string 'Test comment'.

    http_request.set_text("GET /addurl?url=http://www.foo.com"
                          "&comment=Test+comment HTTP/1.1\r\n\r\n");

    // Tell the FakeAddUrlService to expect to receive a URL
    // of 'http://www.foo.com' again.

    fake_add_url_service_->set_expected_url("http://www.foo.com");

    // Tell the FakeAddUrlService to also expect to receive a
    // comment of 'Test comment' this time.

    fake_add_url_service_->set_expected_comment("Test comment");
}

```

Разработчик напишет еще много похожих тестов, но этот пример хорошо демонстрирует общую схему определения имитации, ее внедрения в тестируемую систему. Он объясняет, как использовать имитацию в тестах для внедрения ошибок и логики проверки в потоке операций тестируемой системы. Один из отсутствующих здесь важных тестов имитирует сетевой тайм-аут между `AddUrlFrontend` и бэкенд-системой `FakeAddUrlService`. Такой тест поможет, если наш разработчик забыл проверить и обработать ситуацию с возникновением тайм-аута.

Знатоки гибкой методологии тестирования укажут, что все функции `FakeAddUrlService` достаточно просты и вместо имитации (`fake`) можно было бы использовать подставной объект (`mock`). И они будут правы. Мы реализовали эти функции в виде имитации исключительно для ознакомления с процессом.

Теперь разработчик хочет выполнить написанные тесты. Для этого он должен обновить свои определения сборки и включить новое тестовое правило, определяющее бинарник теста `addurl_frontend_test`.

FILE: DEPOT/ADDURL/BUILD

```
# From before:  
proto_library(name="addurl",  
    srcs=["addurl.proto"])  
  
# Also from before:  
cc_library(name="addurl_frontend",  
    srcs=["addurl_frontend.cc"],  
    deps=[  
        "path/to/httpqueryparams",  
        "other_http_server_stuff",  
        ":addurl", # Depends on the proto_library above.  
])  
# New:  
cc_test(name="addurl_frontend_test",  
    size="small", # See section on Test Sizes.  
    srcs=["addurl_frontend_test.cc"],  
    deps=[  
        ":addurl_frontend", # Depends on library above.  
        "path/to/googletest_main"])
```

И снова разработчик использует свои инструменты сборки для компилирования и запуска бинарного файла `addurl_frontend_test`, исправляет все обнаруженные ошибки компилятора и компоновщика. Кроме того, он исправляет тесты, тестовые фикстуры, имитации и саму `AddUrlFrontend` по всем падениям тестов. Этот процесс начинается сразу же после определения `FixtureTest` и повторяется при следующих добавлениях тестовых сценариев. Когда все тесты готовы и успешно проходят, разработчик создает список изменений, содержащий все файлы, а заодно исправляет все мелкие проблемы, выявленные в ходе предварительных проверок. После этого он отправляет список изменений на рецензирование и переходит к следующей задаче (скорее всего, начинает писать реальный бэкенд `AddUrlService`), одновременно ожидая обратной связи от рецензента.

```
$ create_cl BUILD \  
    addurl.proto \  
    addurl_frontend.h \  
    addurl_frontend.cc \  
    addurl_frontend_test.cc  
$ mail_cl -m reviewer@google.com
```

Получив обратную связь, разработчик вносит соответствующие изменения или вместе с рецензентом находит альтернативные решения, возможно — проходит дополнительное рецензирование, после чего отправляет список изменений в систему контроля версий. Системы автоматизации тестирования Google знают, что начиная с этого момента при внесении изменений в код, содержащийся в этих файлах, следует выполнить `addurl_frontend_test` и убедиться, что новые изме-

нения не ломают существующие тесты. Каждый разработчик, который собирается изменять `addurl_frontend.cc`, может использовать `addurl_frontend_test` как страховку для внесения изменений.

Выполнение тестов

Автоматизация тестирования — это больше, чем просто написание отдельных тестов. Если подумать, что еще нужно для хорошего результата, мы увидим, что в автоматизации не обойтись без компиляции тестов и их выполнения, анализа, сортировки и формирования отчетов о результатах каждого прогона. Автоматизация тестирования — это полноценная разработка ПО со всеми вытекающими.

Вся эта работа мешает инженерам сосредоточиться на сути — написании правильных автотестов, приносящих пользу проекту. Код тестов полезен настолько, насколько он ускоряет процесс разработки. Чтобы этого достичь, его нужно встраивать в процесс разработки основного продукта так, чтобы он стал его естественной частью, а не побочной деятельностью. Код продукта никогда не существует в вакууме, сам по себе. Так же должно быть и с кодом тестов.

Вот почему мы построили общую инфраструктуру, которая отвечает за компиляцию, прогон, анализ, хранение и отчетность о тестах. Внимание инженеров Google вернулось к написанию отдельных тестов. Они просто отправляют их в эту общую инфраструктуру, которая заботится о выполнении тестов и следит, чтобы тестовый код обслуживался так же, как и функциональный.

Написав новый набор тестов, разработчик в тестировании создает спецификацию на сборку этого теста для нашей инфраструктуры сборки. Спецификация на сборку теста содержит название теста, исходные файлы для сборки, зависимости файлов от прочих библиотек и данных и, наконец, размер теста. Размер задается обязательно для каждого теста: малый, средний, большой или громадный. Человек только заливает код тестов и спецификацию сборки в систему, средства сборки и инфраструктура прогона тестов Google берут на себя все остальное. Всего лишь по одной команде запустится сборка, выполнится автотест и покажутся результаты этого прогона.

Инфраструктура выполнения тестов накладывает на тесты некоторые ограничения. Что это за ограничения и как с ними работать, мы расскажем в следующем разделе.

Определения размеров тестов

По мере роста Google и прихода новых сотрудников в компании началась путаница с названиями тестов: юнит-тесты, тесты на основе кода, тесты белого ящика, интеграционные тесты, системные тесты и сквозные тесты — все они выделяли разные уровни детализации, как рассказывает Пэм на рис. 2.1. Однажды мы решили, что так дальше продолжаться не может, и создали стандартный набор типов тестов.

Малые тесты проверяют работу каждой единицы кода независимо от ее окружения. Примеры таких единиц кода: отдельные классы или небольшие группы связанных функций. У малых тестов не должно быть внешних зависимостей. Вне Google такие малые тесты обычно называют юнит-тестами.

У малых тестов самый узкий охват, и они фокусируются на одной, отделенной от всего, функции, как показано на рис. 2.2 на следующей странице. Такой узкий охват малых тестов позволяет им обеспечивать исчерпывающее покрытие низкоуровневого кода, недоступное для более крупных тестов.

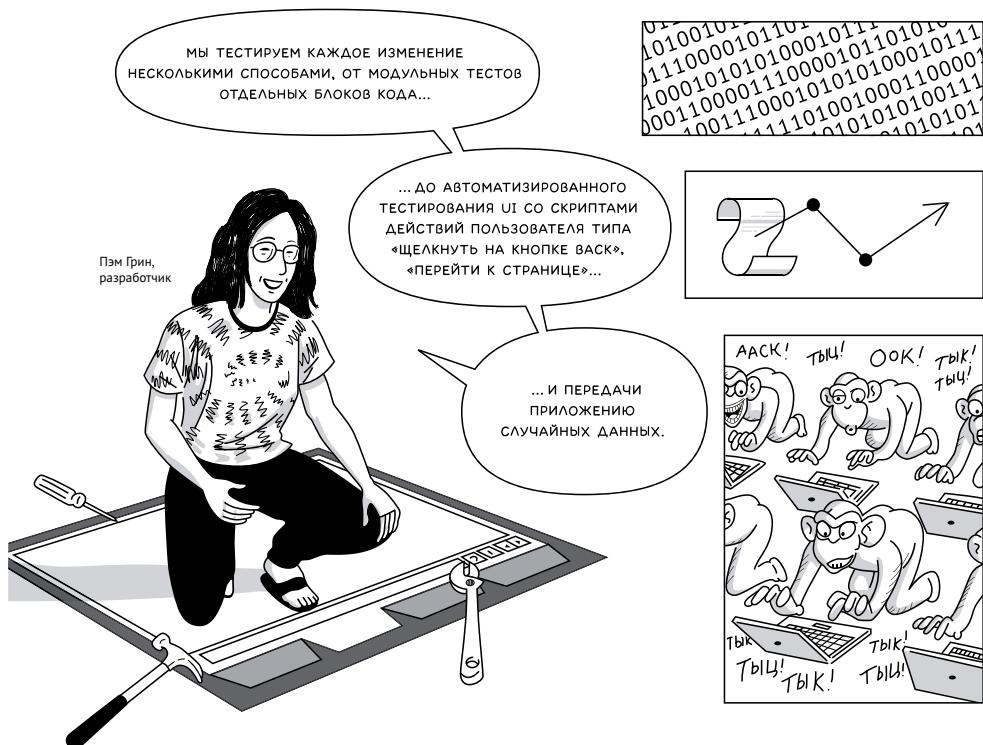


Рис. 2.1. В Google используется много разных видов тестов

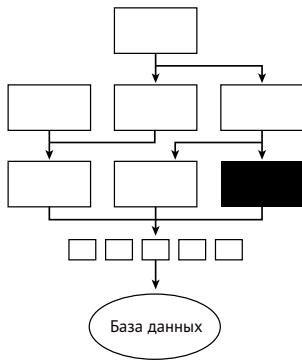


Рис. 2.2. В малом тесте обычно проверяется всего одна функция

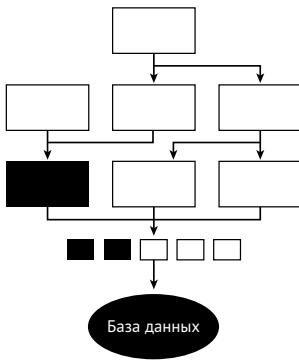


Рис. 2.3. Средние тесты охватывают несколько модулей и могут задействовать внешние источники данных

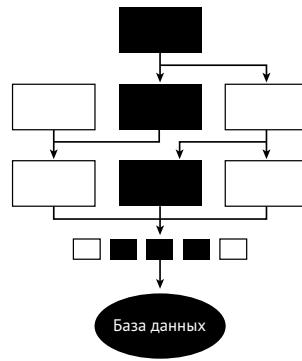


Рис. 2.4. Большие и громадные тесты включают модули, необходимые для сквозного выполнения задач

Для малых тестов необходимо имитировать внешние сервисы вроде файловой системы, сетей и баз данных через подставные объекты и имитации. Лучше имитировать даже внутренние сервисы, которые находятся внутри того же модуля, что и тестируемый класс. Чем меньше внешних зависимостей — тем лучше для малых тестов.

Ограниченный охват и отсутствие внешних зависимостей означают, что малые тесты могут работать очень быстро. Следовательно, их можно часто запускать и быстро находить ошибки. Задумка в том, чтобы разработчик, по мере запуска тестов и правки основного кода, заодно отвечал за поддержку этого тестового кода. Изоляция малых тестов также позволяет сократить время их сборки и исполнения.

Средние тесты проверяют взаимодействие между двумя или более модулями приложения, как это показано на рис. 2.3. Средние тесты отличаются от малых большим охватом и временем прогона. Если малые тесты пытаются задействовать весь код одной функции, средние тесты направлены на взаимодействие между определенным набором модулей. За пределами Google такие тесты обычно называют интеграционными.

Средними тестами нужно управлять через тестовую инфраструктуру. Из-за большего времени прогона они запускаются реже. В основном эти тесты создаются и выполняются силами разработчиков в тестировании.

НА ЗАМЕТКУ

Малые тесты проверяют поведение отдельной единицы кода. Средние тесты проверяют взаимодействие одного или нескольких модулей кода. Большие тесты проверяют работоспособность системы в целом.

Имитация внешних сервисов для средних тестов приветствуется, но не обязательна. Это может быть полезно, если нужно увеличить быстродействие. Там, где полноценная имитация сервисов неоправданна, для повышения производительности можно использовать облегченные вариации, например встроенные в память базы данных.

Большие и громадные тесты за пределами Google называют системными тестами, или сквозными тестами. Большие тесты оперируют на высоком уровне и проверяют, что система работает как единое целое. Эти тесты задействуют все подсистемы, начиная с пользовательского интерфейса и заканчивая хранилищами данных, как это показано на рис. 2.4. Они могут обращаться к внешним ресурсам, таким как базы данных, файловые системы и сетевые службы.

Как мы используем размеры тестов в общей инфраструктуре

Автоматизацию тестирования трудно сделать универсальной. Чтобы все проекты в большой ИТ-компании могли работать с общей тестовой инфраструктурой, она должна поддерживать множество разных сценариев запуска тестов.

Например, вот некоторые типичные сценарии запуска тестов, которые поддерживает общая инфраструктура тестирования Google.

- Разработчик хочет скомпилировать и запустить малый тест и тут же получить результаты.
- Разработчик хочет запустить все малые тесты для проекта и тут же получить результаты.
- Разработчик хочет скомпилировать и запустить только те тесты, которые связаны с последним изменением кода, и тут же получить результаты.
- Разработчик или тестировщик хочет собрать данные о покрытии кода в конкретном проекте и посмотреть результаты.
- Команда хочет прогонять все малые тесты для своего проекта каждый раз при создании списка изменений и рассыпать результаты всем участникам команды.
- Команда хочет прогонять все тесты для своего проекта после отправки списка изменений в систему управления версиями.
- Команда хочет еженедельно собирать статистику о покрытии кода и отслеживать его прогресс со временем.

Может быть и так, что все вышеперечисленные задания отправляются в систему выполнения тестов Google одновременно. Некоторые из тестов могут захватывать ресурсы, занимая общие машины на целые часы. Другим будет достаточно миллисекунд для выполнения, и они могут благополучно исполняться на одной машине с сотнями других тестов. Когда тесты помечены как малые, средние и большие, гораздо проще планировать расписание выполнения запусков, так как планировщик понимает, сколько времени может занять запуск, и оптимизирует очередь.

Система выполнения тестов Google отличает быстрые задания от медленных по информации о размере тестов. У каждого размера есть верхняя граница времени выполнения теста (табл. 2.1). Размер определяет и потенциальную потребность в ресурсах (табл. 2.2). Система прерывает выполнение и сообщает об ошибке, если тест превышает выделенное для его категории время или доступный объем ресурса. Это мотивирует разработчиков в тестировании назначать правильные метки размеров тестов. Точное определение размеров тестов позволяет системе строить эффективное расписание.

	Малые тесты	Средние тесты	Большие тесты	Огромные тесты
Временные цели (на метод)	Исполнение меньше чем за 100 мс	Исполнение меньше чем за 1 с	Как можно быстрее	Как можно быстрее
Ограничения по времени	Прерывание малых тестов после 1 минуты	Прерывание средних тестов после 5 минут	Прерывание больших тестов после 15 минут	Прерывание огромных тестов после 1 часа

Таблица 2.1. Цели и ограничения времени отработки тестов по их размеру

Ресурс	Большие	Средние	Малые
Сетевые службы (открытие сокета)	Да	Только localhost	Подставные объекты
База данных	Да	Да	Подставные объекты
Доступ к файловой системе	Да	Да	Подставные объекты
Доступ к пользовательским системам	Да	Нежелательно	Подставные объекты
Вызов системных функций	Да	Нежелательно	Подставные объекты
Многопоточность	Да	Да	Нежелательно
Команды приостановки	Да	Да	Нет
Свойства системы	Да	Да	Нет

Таблица 2.2. Использование ресурсов в зависимости от размеров теста

Преимущества разных размеров тестов

Размер теста имеет значение. Он влияет на специфические преимущества теста. На рис. 2.5 показана общая сводка, а ниже мы приводим более подробный список достоинств и недостатков каждого типа тестов.



Рис. 2.5. Ограничения разных размеров тестов

Большие тесты

Достоинства и недостатки больших тестов:

- Большие тесты проверяют самое важное — работу приложения. Они учитывают поведение внешних подсистем.
- Большие тесты могут быть недетерминированными (результат может быть получен разными путями), потому что зависят от внешних подсистем.
- Большой охват усложняет поиск причин при неудачном прохождении теста.
- Подготовка данных для тестовых сценариев может занимать много времени.
- Из-за высокоуровневости больших тестов в них трудно прорабатывать граничные значения. Для этого нужны малые тесты.

Средние тесты

Достоинства и недостатки средних тестов:

- Требования к подставным объектам мягче, а временные ограничения свободнее, чем у малых тестов. Разработчики используют их как промежуточную ступень для перехода от больших тестов к малым.
- Средние тесты выполняются относительно быстро, поэтому разработчики могут запускать их часто.

- Средние тесты выполняются в стандартной среде разработки, поэтому их очень легко запускать.
- Средние тесты учитывают поведение внешних подсистем.
- Средние тесты могут быть недетерминированными, потому что зависят от внешних подсистем.
- Средние тесты выполняются не так быстро, как малые.

Малые тесты

Достоинства и недостатки малых тестов:

- Малые тесты помогают повысить чистоту кода, потому что работают узко-направленно с небольшими методами. Соблюдение требований подставных объектов приводит к хорошо структурированным интерфейсам между подсистемами.
- Из-за скорости выполнения малые тесты выявляют баги очень рано и дают немедленную обратную связь при внесении изменений в код.
- Малые тесты надежно выполняются во всех средах.
- Малые тесты обладают большей детализацией, а это упрощает тестирование граничных случаев и поиск состояний, приводящих к ошибкам, например `null`-указатели.
- Узкая направленность малых тестов сильно упрощает локализацию ошибок.
- Малые тесты не проверяют интеграцию между модулями — для этого используются другие тесты.
- Иногда сложно применить подставные объекты для подсистем.
- Подставные объекты и псевдосреды могут отличаться от реальности.

Малые тесты способствуют созданию качественного кода, хорошей проработке исключений и получению информации об ошибках. Более масштабные тесты ориентированы на общее качество продукта и проверку данных. Ни один тип тестов не покрывает все потребности продукта в тестировании. Поэтому в проектах Google мы стараемся использовать разумное сочетание всех типов тестов в каждом тестовом наборе. Автоматизация, основанная только на больших комплексных тестах, так же вредна, как и создание только малых юнит-тестов.

НА ЗАМЕТКУ

Малые тесты направлены на проверку качества кода, а средние и большие — на проверку качества всего продукта.

Покрытие кода — отличный инструмент, чтобы оценить, насколько разумно используется сочетание разных размеров тестов в проекте. Проект генерирует один отчет с данными покрытия только для малых тестов, а потом другой отчет с данными только для средних и больших тестов. Каждый отчет в отдельности должен показывать приемлемую величину покрытия для проекта. Если средние и большие тесты в отдельности обеспечивают только 20-процентное покрытие, а покрытие малыми тестами приближается к 100, то у проекта не будет доказательств работоспособности всей системы. А если поменять эти числа местами, скорее всего, расширение или сопровождение проекта потребует серьезных затрат на отладку. Чтобы генерировать и просматривать данные о покрытии кода на ходу, мы используем те же инструменты, которые собирают и выполняют тесты. Достаточно поставить дополнительный флаг в командной строке. Данные о покрытии кода хранятся в облаке, и любой инженер может просмотреть их через веб в любой момент.

Google разрабатывает самые разные проекты, их потребности в тестировании сильно отличаются. В начале работы мы обычно используем правило 70/20/10: 70% малых тестов, 20% — средних и 10% — больших. В пользовательских проектах со сложными интерфейсами или высокой степенью интеграции доля средних и крупных тестов должна быть выше. В инфраструктурных проектах или проектах, где много обработки данных (например, индексирование или обход веб-контента), малых тестов нужно намного больше, чем больших и средних.

Для наблюдения за покрытием кода в Google используется внутренний инструмент — Harvester. Это инструмент визуализации, который отслеживает все списки изменений проекта и графически отображает важные показатели: отношение объема кода тестов к объему нового кода в конкретных списках изменений; размер изменений; зависимость частоты изменений от времени и даты; распределение изменений по разработчикам и т. д. Цель Harvester — дать общую сводку об изменениях в процессе тестирования проекта со временем.

Требования к выполнению тестов

У системы выполнения тестов в Google одинаковые требования ко всем тестам.

- Каждый тест должен быть независим от других, чтобы тесты могли выполняться в любом порядке.
- Тесты не должны иметь долгосрочных последствий. После их завершения среда должна возвращаться в то же состояние, в котором она находилась при запуске.

Требования простые и понятные, но выполнить их оказывается не так просто. Даже если сам тест отвечает требованиям, тестируемая программа может их нарушать, сохраняя файлы данных или изменяя конфигурацию. К счастью, сама среда выполнения тестов Google упрощает соблюдение этих требований.

Что касается требования независимости, инженер во время прогона может установить флаг выполнения тестов в случайном порядке. Эта фича помогает выявить зависимости, связанные с порядком выполнения. Впрочем, случайный порядок может означать, что тесты запускаются параллельно. Система может отправить выполнять два теста на одной машине. Если каждый тест требует единоличного доступа к ресурсам системы, один из них упадет. Например:

- оба теста пытаются подключиться к одному порту для единоличного получения сетевого трафика;
- оба теста пытаются создать каталог, используя один путь;
- один тест создает и заполняет таблицу базы данных, а другой пытается удалить ту же таблицу.

Такие конфликты могут вызывать сбои не только в самих тестах, но и в соседних тестах, которые выполняются в той же системе, даже если эти другие тесты соблюдают правила. Наша система умеет выявлять такие ситуации и оповещать владельцев тестов-бунтарей.

Если установить специальный флаг, тест будет выполняться единолично на выделенной машине. Но это лишь временное решение. Все равно придется переписать тесты и удалить зависимости от критических ресурсов. Например, эти проблемы можно решить так:

- каждый тест запрашивает свободный порт у системы выполнения тестов, а тестируемая программа динамически к нему подключается;
- каждый тест создает все папки и файлы во временной директории, созданной и выделенной системой специально для него перед выполнением тестов;
- каждый тест работает со своим экземпляром базы данных в изолированной среде с выделенными системой выполнения тестов директориями и портами.

Ребята, ответственные за сопровождение системы выполнения тестов Google, довольно подробно описали свою среду выполнения тестов. Их документ называется «Энциклопедией тестирования Google», и он отвечает на все вопросы о том, какие ресурсы доступны тестам во время выполнения. «Энциклопедия тестирования» составлена как стандартизированный документ, где у терминов «должен» и «будет» однозначное значение. В энциклопедии подробно объясняются роли и обязанности тестов, исполнителей тестов, систем хостинга, рантайм-библиотек, файловых систем и т. д.

Вряд ли все инженеры Google читали «Энциклопедию тестирования». Скорее всего, большинство предпочитает учиться у других, или испытывать метод проб и ошибок, или постоянно натыкаться на комментарии рецензентов их кода. Они и не подозревают, что общая среда выполнения тестов может обслужить все проекты по тестированию Google. Чтобы это узнать, достаточно заглянуть в энциклопедию. Им неизвестно, что этот документ — главная причина того, что тесты ведут себя в общей среде ровно так же, как и на личной машине написавшего тест инженера. Технические детали даже самых сложных систем остаются незамеченными теми, кто их использует. Все же работает, зачем читать.

Тестирование на скоростях и в масштабах Google

Пуджа Гупта, Марк Айви и Джон Пеникс

Системы непрерывной интеграции — главные герои обеспечения работоспособности программного продукта во время разработки. Типичная схема работы большинства систем непрерывной интеграции такая.

1. Получить последнюю копию кода.
2. Выполнить все тесты.
3. Сообщить о результатах.
4. Перейти к пункту 1.

Решение отлично справляется с небольшой кодовой базой, пока динамичность изменений кода не выходит за рамки, а тесты прогоняются быстро. Чем больше становится кода, тем сильнее падает эффективность подобных систем. Добавление нового кода увеличивает время «чистого» запуска, и в один прогон включается все больше изменений. Если что-то сломается, найти и исправить изменение становится все сложнее.

Разработка программных продуктов в Google происходит быстро и с размахом. Мы добавляем в базу кода всего Google больше 20 изменений в минуту, и 50% файлов в ней меняются каждый месяц. Разработка и выпуск всех продуктов опираются на автотесты, проверяющие поведение продукта. Есть продукты, которые выпускаются несколько раз в день, другие — раз в несколько недель.

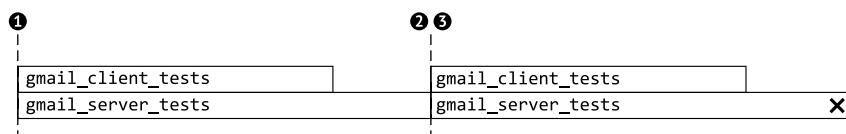
По идеи, при такой огромной и динамичной базе кода команды должны тратить кучу времени только на поддержание сборки в состоянии «зелено-

го света». Система непрерывной интеграции должна помочь с этим. Она должна сразу выделять изменение, приводящее к сбою теста, а не просто указывать на набор подозрительных изменений или, что еще хуже, перебирать их все в поисках нарушителя.

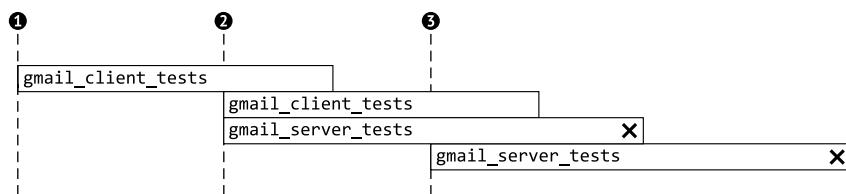
Чтобы решить эту проблему, мы построили систему непрерывной сборки (рис. 2.6), которая анализирует зависимости и выделяет только те тесты, которые связаны с конкретным изменением, а потом выполняет только их. И так для каждого изменения. Система построена на инфраструктуре облачных вычислений Google, которая позволяет одновременно выполнять большое количество сборок и запускать затронутые тесты сразу же после отправки изменений.

Примером ниже мы показываем, как наша система дает более быструю и точную обратную связь, чем типичная непрерывная сборка. В нашем сценарии используются два теста и три изменения, затрагивающие эти тесты. Тест `gmail_server_tests` падает из-за изменения 2. Типичная система непрерывной сборки сообщила бы, что к сбою случился из-за изменения 2 или 3, не уточняя. Мы же используем механизм параллельного выполнения, поэтому запускаем тесты независимо, не дожидаясь завершения текущего цикла «сборка–тестирование». Анализ зависимостей сузит набор тестов для каждого изменения, поэтому в нашем примере общее количество выполнений теста то же самое.

Типичная система непрерывной интеграции



Система непрерывной интеграции с анализом зависимостей



— -- Изменение инициирует выполнение тестов

Тесты выполняются. Длина прямоугольника обозначает время выполнения теста

Тест не прошел

Rис. 2.6. Сравнение систем непрерывной интеграции

Наша система берет данные о зависимостях из спецификаций сборки, которые описывают, как компилируется код и какие файлы входят в сборку приложения и теста. Правила сборки имеют четкие входные и выходные данные, объединив которые получим точное описание процесса сборки. Наша система строит в памяти график зависимостей сборки, как на рис. 2.7, и обновляет его с каждым новым изменением. На основании этой схемы мы определяем все тесты, связанные прямо или косвенно с кодом, вошедшим в изменение. Именно эти тесты нужно запустить, чтобы узнать текущее состояние сборки. Давайте посмотрим на пример.

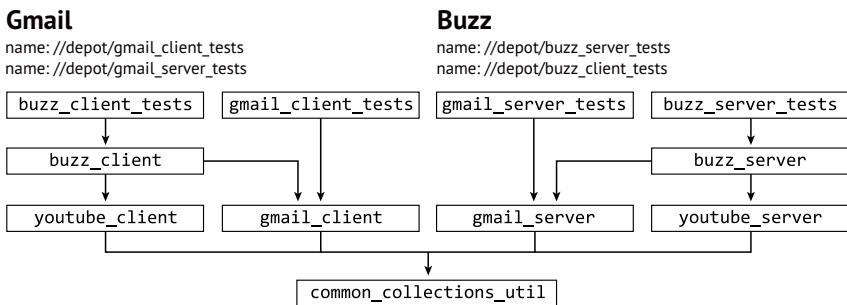


Рис. 2.7. Пример зависимостей сборки

Мы видим, как два отдельных изменения в коде, находящихся на разных уровнях дерева зависимостей, анализируются, чтобы подобрать минимальный набор тестов, который определит, дать ли зеленый свет проектам Gmail и Buzz.

Сценарий 1: изменение в общей библиотеке

Для первого сценария возьмем изменение, которое модифицирует файлы в `common_collections_util`, как показано на рис. 2.8.

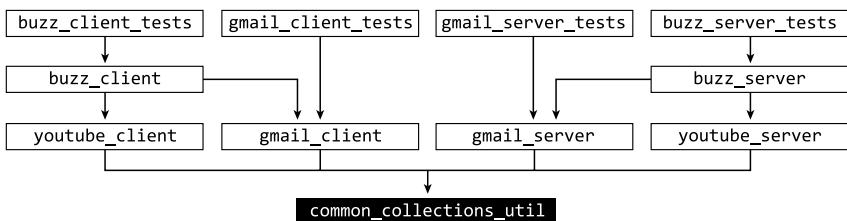


Рис. 2.8. Изменение в common_collections_util.h

Отправив изменение, мы перемещаемся по линиям зависимостей вверх по графику. Так мы найдем все тесты, зависящие от изменений. Когда поиск завершится, а это займет лишь доли секунды, у нас будут все тесты, которые нужно прогнать, и мы получим актуальные статусы наших проектов (рис. 2.9).

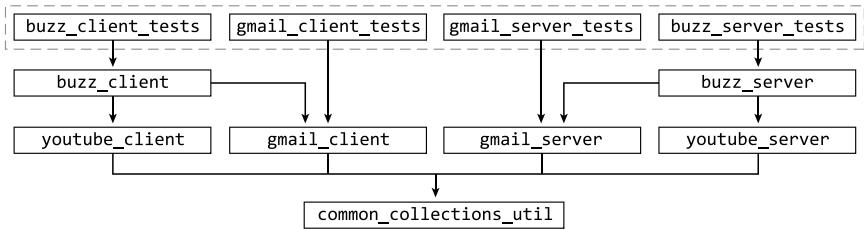


Рис. 2.9. Тесты, на которые влияет изменение

Сценарий 2: изменение в зависимом проекте

Во втором примере возьмем изменение, которое модифицирует файлы в youtube_client (рис. 2.10).

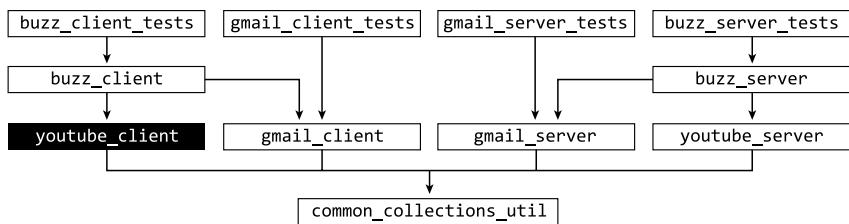


Рис. 2.10. Изменение в youtube_client

Проведя аналогичный анализ, мы определим, что изменение влияет только на buzz_client_tests и что нужно актуализировать статус проекта Buzz (рис. 2.11).

Запускаются только тесты buzz_client_tests,
и только проект Buzz нуждается в обновлении

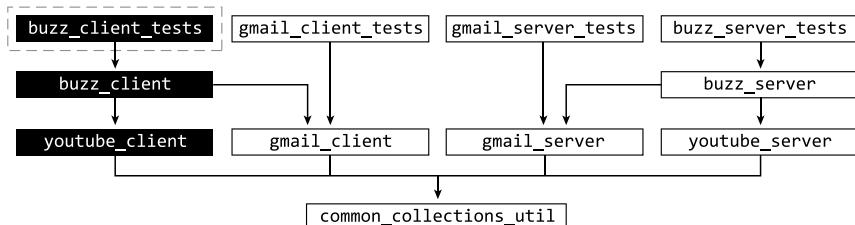


Рис. 2.11. Buzz нужно обновить

Примеры показывают, как мы оптимизируем количество тестов, прогоноемых для одного изменения, без потери в точности результатов. Уменьшение количества тестов для одного изменения позволяет выполнить все нужные тесты для каждого зафиксированного изменения. Нам становится легче выявлять и отлаживать проблемы в проблемном изменении.

Умные инструменты и возможности инфраструктуры облачных вычислений сделали систему непрерывной интеграции быстрой и надежной. И мы постоянно стараемся ее улучшить, хотя она уже используется в тысячах проектов Google, чтобы выпускать проекты быстрее и проводить больше итераций. И — что важно — наш прогресс замечают пользователи.

Тест-сертификация

В начале книги Патрик Коупленд замечает, как сложно было привлечь разработчиков к тестированию. Первым делом мы создали им отличную компанию и наняли технически подкованных тестировщиков. А чтобы втянуть разработчиков в процесс, мы придумали «Тест-сертификацию». Оглядываясь назад, можно сказать, эта программа сыграла важную роль в становлении культуры тестирования разработчиками в Google.

Тест-сертификация начиналась как соревнование. Будут ли разработчики серьезно относиться к тестированию, если мы сделаем эту работу престижной? Что, если награждать разработчиков, которые следуют тестовым практикам? А что, если мы скажем, что они теперь сертифицированные инженеры? А может, еще ввести систему наградных бейджей (рис. 2.12), которыми можно пощеголять перед коллегами?



Рис. 2.12. Бейджи тест-сертификации показываются на вики-страницах проектов

Мы изобрели тест-сертификацию — это система заданий по тестированию, которые должна выполнить команда, чтобы стать сертифицированной. Все команды начинают с нулевого уровня. Если команда показывает мастерство базовой гигиены кода, ейдается первый уровень. Уровень команды постепенно растет с тем, как она учится писать все более чистый код. В игре в сертификацию всего пять уровней, как и во многих серьезных моделях зрелости разработки ПО.

Краткое описание уровней Тест-сертификации

Уровень 1

- Создать пакеты тестового покрытия.
- Установить систему непрерывной сборки.
- Ранжировать тесты на малые, средние и большие.
- Определить недетерминированные тесты.
- Создать набор смоук-тестов.

Уровень 2

- Не выпускать, пока не пройдут все тесты.
- Обязательно выполнять смоук-тесты до отправки кода.
- Инкрементальное покрытие всеми тестами не меньше 50%.
- Инкрементальное покрытие малыми тестами не меньше 10%.
- Хотя бы одна фича покрыта интеграционным тестом.

Уровень 3

- Создавать тесты для всех нетривиальных изменений
- Общее покрытие малыми тестами не меньше 50%.
- Важные новые фичи покрыты интеграционными тестами.

Уровень 4

- Смоук-тесты запускаются автоматически перед отправкой нового кода.
- Смоук-тесты проходят за время меньше 30 минут.
- Нет недетерминированных тестов.
- Общее тестовое покрытие не меньше 40%.
- Тестовое покрытие только малыми тестами не меньше 25%.
- Все важные фичи покрыты интеграционными тестами.

Уровень 5

- Добавить тест к исправлению всех нетривиальных багов.
- Активно использовать доступные средства анализа.
- Общее тестовое покрытие не меньше 60%.
- Тестовое покрытие только малыми тестами не меньше 40%.

Сначала мы обкатали программу на нескольких командах разработчиков, которые и так были позитивно настроены по отношению к тестированию. Они хотели улучшить свои навыки. Когда мы сбалансировали механизм наград, мы объявили открытым большое соревнование за бейджи по всей компании. Наша программа была принята на ура.

Продать нашу идею оказалось не так сложно, как казалось. Команды разработки только выигрывали от этого:

- Они получали поддержку от опытных тестировщиков, которые согласились стать *наставниками* тест-сертификации. Ресурсы тестирования всегда в дефиците, а присоединившись к программе, команда получала больше тестировщиков, чем ей было положено официально.
- Они получали помощь экспертов и учились лучше писать малые тесты.
- Они видели, какие команды лучше проводят тестирование, и понимали, у кого стоило учиться.
- Они могли похвастаться перед другими командами своим уровнем тест-сертификации.

Вся компания могла следить, как команды разработки зарабатывают уровни.

Руководители разработки успешных команд получали хорошие отзывы от руководителей направления продуктивности разработки. Команды, которые выражали скептицизм и подтрунивали над участвующими, на самом деле рисковали. Если ресурсы тестирования так сложно получить, зачем лишний раз разочаровывать ребят из направления продуктивности? Конечно, не все шло гладко. Дадим слово людям, которые руководили этой программой.

Интервью с основателями программы тест-сертификации

Марк Стрибек — руководитель разработки Gmail. Нил Норвиц — разработчик, занимающийся инструментами ускорения разработки. Трэйси Бялик и Росс Руфер — разработчики в тестировании, одни из лучших инженеров Google. Эти четверо помогали запустить программу тест-сертификации.

Расскажите, с чего началась тест-сертификация? Какие проблемы команда пыталась решить при запуске? Совпадают ли они с теми, которые вы решаете сегодня?

Трэйси: Мы хотели изменить культуру разработчиков Google, чтобы тестирование стало ежедневной обязанностью каждого программиста. Вначале мы просто

делились положительным опытом в тестировании и уговаривали команды писать тесты. Некоторые заинтересовались, но не могли решить, с чего начать. Другие включали совершенствование тестирования в список квартальных целей. Обычно это так и оставалось только на бумаге, как постоянные обещания себе похудеть в следующем году. Хорошая, достойная цель, но если ее формулировать абстрактно, не удивляйтесь, если она так и не сможет обрести плоть.

Программа тест-сертификации помогает спланировать совершенствование тестирования по шагам. На первом уровне мы формируем основу: настраиваем фреймворк для запуска автотестов и собираем данные о тестовом покрытии, выявляем недетерминированные тесты и создаем пакет смок-тестов, если полный тестовый пакет выполняется долго.

Постепенно уровни усложняются, требуя все большей прокачки в тестировании. На втором уровне нужно определяться со стратегией и начинать улучшать инкрементальное покрытие. Третий уровень направлен на тестирование нового кода. На четвертом уровне тестируется старый код, который раньше не тестировался. Может быть, придется провести рефакторинг, чтобы код стал тестируемым. На пятом уровне уже нужно иметь хорошее общее покрытие, писать тесты для всех исправленных багов и использовать средства для статического и динамического анализа.

Теперь все сотрудники Google знают, что тестирование входит в обязанности разработчиков. Одной проблемой меньше. Но у нас остается задача помочь командам повысить навыки тестирования и научить их новым методам. Так что сейчас тест-сертификация направлена именно на это.

— Как оценивали разработчики тест-сертификацию на старте?

Нил: Они считали, что программа чересчур сложна, а наши цели завышены, ведь многим командам нужно было еще добраться даже до начальной стадии. Мы должны были настроить уровни так, чтобы на их достижение у людей было время. Кроме того, инструментарий Google тогда был недоработан и некоторые наши требования превосходили возможности. Людям было трудно влиться в процесс, поэтому нам пришлось подумать о том, как смягчить требования на старте и убедить команды в том, что они действительно продвигаются вперед.

Марк: Да, нам пришлось отступить на несколько шагов назад. Мы постарались стать более реалистичными и увеличить длину взлетной полосы. Тем не менее, каким бы длинным ни был разгон, все равно для взлета надо развить порядочную скорость. Так мы сформулировали первый шаг: настроить непрерывную сборку, доводить хотя бы некоторые сборки до состояния «зеленый свет», определить покрытие кода. Это и так было у многих команд. Когда мы оформили это в правила, команды смогли сразу перейти на первый уровень и уже хотели работать дальше по программе.

— А кто принял вашу идею с энтузиазмом?

Нил: У нас организовалась группа ребят, заинтересованных в тестировании. Мы регулярно собирались. Потом мы стали приглашать знакомых коллег. Приятным сюрпризом оказалось, что многим инженерам это интересно. Интерес к программе вырос, когда мы стали применять «тестирование в туалете»¹ и другие штуки, которые делали тестирование более прикольным: «починялки»², массовая рассылка электронной почты, плакаты, выступления на пятничных встречах «Thanks God It's Friday» и пр.

Марк: Как только мы обратились к другим командам, а заинтересованных было немало, они поняли, что нужна серьезная работа, для которой у них просто нет нужного опыта. Начало было невеселым.

— Кто не хотел поддерживать ваше нововведение?

Нил: Большинство проектов. Как я уже сказал, всем казалось, что это слишком сложно. Мы должны были умерить свои начальные амбиции. По сути, в Google было два вида проектов: без тестов и с очень скверными тестами. Мы должны были показать тестирование как обычную рутину, с которой можно справиться еще до обеда. И это было реально — с нашей помощью.

Марк: Кроме того, в то время в Google еще не было единого понимания ценности тестирования и автоматизации. Сегодня все по-другому. А тогда большинство команд полагало, что все эти идеи очень милые, но у них есть дела поважнее, а именно — писать код.

— Какие препятствия пришлось преодолеть, чтобы собрать единомышленников?

Нил: Инертность. Плохие тесты. Отсутствие тестов. Время. Тестирование рассматривалось как проблема кого-то другого — не важно, другого разработчика или тестировщика. Когда заниматься тестами, если нужно написать столько кода?

Марк: Пришлось искать команды, которые были бы достаточно заинтересованными и не имели особых проблем со старым кодом. В них должен был быть хотя бы один человек, который хотел заниматься тестированием и ориентировался

1 «Тестирование в туалете» (Testing on The Toilet) упоминалось ранее в этой книге. О нем часто пишут в блоге Google Testing по адресу googletesting.blogspot.com

2 «Починялки» (fixits) — еще один элемент культуры Google: это мероприятие собирает людей для починки чего-то сломанного или работающего неправильно. Например, группа может устроить «починялки» для сокращения количества ошибок. Или для тестирования безопасности. Или для расширения использования `#include` в коде C или рефакторинга. Цель встречи не ограничивается технической областью, и «починялки» могут проводиться для улучшения качества блюд в кафе или схемы проведения собраний. По сути, это любое мероприятие, на котором люди собираются для решения общей проблемы.

в теме. Это были три основных препятствия, и мы последовательно преодолевали их, команда за командой.

— Как тест-сертификация стала популярной? Рост был вирусный или линейный?

Расс: Сначала мы прогнали пилотную версию на командах, которые дружили с тестированием, и на тех, кто помогал организовать программу. Мы тщательно выбирали свою аудиторию — привлекали команды с самыми высокими шансами на успех.

Когда мы объявили глобальный запуск тестовой сертификации в середине 2007 года, в Google уже было 15 пилотных команд, находящихся на разных уровнях. Перед запуском мы обклеили стены всех наших зданий в Маунтин-Вью плакатами «Что скрывает тест-сертификация». На плакатах были фотографии команд и внутренние названия проектов (Rubix, Bounty, Mondrian, Red Tape и т. д.). Мы сделали провокационные надписи типа «Будущее начинается сегодня» и «Это важно, не отставай» со ссылкой на программу. Мы получили огромное количество посещений от любопытствующих, которые хотели раскрыть тайну или подтвердить свои догадки. Мы задействовали «туалетное тестирование», чтобы прорекламировать программу и рассказать людям, где они могут узнать больше о проекте.

Мы объясняли, почему программа важна для команд и чем она им поможет. Мы подчеркивали, что команды получат наставника по тест-сертификации и доступ к сообществу экспертов тестирования. За участие в проекте команды получали два подарка. Первый — светящийся шар «статуса сборки», который показывал командам состояние тестов их непрерывных сборок в цвете: тесты прошли успешно — зеленый, не прошли — красный. Второй — комплект прикольных фигурок из «Звездных войн», который мы называли «Дарт Тестер». Всего было три фигурки, и мы выдавали командам по одной по мере продвижения. Команды, на чьих столах появлялись такие шары и фигурки, вызывали любопытство и разговоры о программе.

Участники нашей ранней группы стали первыми наставниками и евангелистами проекта. Постепенно к нам присоединялись новые команды. В них находились свои инженеры-энтузиасты, которые помогали создать шумиху вокруг тест-сертификации и сами становились наставниками.

Чем больше новых команд присоединялось к программе, тем лучше мы учились находить подходящие аргументы для каждой. Для одних команд решающим было то, что пошаговый процесс и наличие наставников поможет им вырасти в данной области. Другие считали, что и сами смогут повысить свою квалификацию, но с официальными уровнями их работа скорее будет оценена по достоинству. Третьи команды уже серьезно использовали методы тестирования, но их можно было уговорить тем, что, присоединившись, они покажут всем, насколько серьезно относятся к тестированию.

Через несколько месяцев, когда в нашей программе уже участвовало порядка 50 команд, несколько смелых представителей от разработки записались в наставники. Это стало началом партнерства между инженерами из команд разработки продуктов и специалистами направления продуктивности.

Весь этот прогресс имел вирусную природу, мы шли снизу, по горизонтали, от сотрудника к сотруднику. Какие-то команды приходилось уговаривать, другие уже приходили к нам сами.

Примерно через год, когда с нами работало уже 100 команд, приток новых сторонников пошел на спад. Тогда Белла Казуэлл, которая занималась привлечением людей, разработала систему поощрений тест-сертификации. За написание новых тестов, привлечение новых команд в проект, совершенствование тестовых методов или достижение новых уровней тест-сертификации начислялись баллы. Появилась индивидуальная система наград. Филиалы по всему миру состязались друг с другом за количество очков. Мы смогли привлечь новых добровольцев, команды и наставников — началась вторая волна популярности программы.

У команд в программе всегда были четкие критерии оценки своего продвижения. К концу 2008 года некоторые руководители стали использовать их для оценки своих команд. Менеджеры из направления продуктивности разработки смотрели на прогресс команды в тест-сертификации, чтобы понять, насколько серьезно они относятся к тестированию и стоит ли им выделять тестировщиков из наших очень ограниченных ресурсов. В некоторых отделах достижение уровня в тест-сертификации стало требованием руководства и критерием для запуска продукта.

К моменту написания этой книги к нам продолжают присоединяться наставники, в программу вступают новые команды, и тест-сертификация прочно укрепилась в компании.

— Как изменилась программа тест-сертификации за первые несколько лет? Изменились ли требования уровней? Изменилась ли система наставничества? Какие изменения стали самыми полезными для участников?

Трэйси: Самым важным изменением стало увеличение числа уровней и пересмотр требований. Сначала у нас было четыре уровня. Перейти с нулевого уровня на первый было легче легкого, мы сделали это намеренно. А вот переход с первого уровня на второй вызывал у большинства трудности, особенно у команд с нетестируемым кодом, доставшимся по наследству. Такие команды быстро перегорали и подумывали бросить программу. Поэтому мы добавили новый уровень между первым и вторым для облегчения перехода. Мы хотели назвать его «Уровень 1,5», но потом решили просто вставить новый уровень и перенумеровать весь список.

Некоторые требования были слишком общими — предписываемое ими соотношение малых/средних/больших тестов не подходило всем командам. Добавив новый уровень, мы обновили эти критерии: удалили отношения размеров тестов, но включили показатели инкрементального покрытия.

Система наставничества все еще существует, правда, теперь у нас много самостоятельных команд. Так как культура тестирования сейчас на подъеме, многим командам уже не нужна активная поддержка. Они хотят только отслеживать свой прогресс. Таким командам мы не назначаем наставника, но отвечаем на их вопросы по почте и присматриваем со стороны за их продвижением по уровням.

Расс: Мы с самого начала помнили, что критерии тест-сертификации нужно применять разумно. Тестирование — это не работа по рецепту. Бывает, что команда не укладывается в общие рамки, на которые мы ориентировались, создавая критерии. Типичные инструменты анализа тестового покрытия или метрики могут не подходить конкретной команде. Каждый критерий имеет под собой обоснование, и мы готовы адаптировать их для команд, не укладывающихся в стереотипы.

— Что даст команде участие в программе тест-сертификации сегодня? Каковы затраты на участие?

Трэйси: Право похвастаться. Четко описанные шаги. Помощь со стороны. Классный светящийся шар. Но настоящая польза — это улучшение тестирования.

Затраты минимальны, если не считать усилий команды на повышение собственного уровня. У нас есть специальное приложение, в котором наставник наблюдает за прогрессом команды и отмечает выполненные шаги. На одной странице есть список всех команд, отсортированный по уровням, со всеми данными по их прогрессу. Можно посмотреть подробную информацию по конкретной команде.

— Есть ли на этих уровнях шаги, которые вызывают больше трудностей, чем другие?

Трэйси: Самый сложный шаг — «обязательные тесты для всех нетривиальных изменений». Когда проект создается с нуля, пишется сразу с расчетом на тестируемость — все просто. С унаследованными проектами, в которых тестирование не учитывалось, могут возникнуть сложности. Иногда нужно писать большой сквозной тест, пытаясь заставить систему пройти конкретные пути кода и отработать определенное поведение, а затем найти, как автоматически собрать результаты. Лучшее, но затратное по времени решение для улучшения тестируемости — рефакторинг кода. Командам, которые пишут код, не учитывая тестирование, потом трудно обеспечить тестовое покрытие, особенно при переходе от малых, узконаправленных юнит-тестов к более крупным тестам, затрагивающим группы классов, не говоря уже о сквозных тестах.

— Многие проекты в Google живут всего несколько недель или месяцев, а ваша тест-сертификация на плаву почти пять лет и, похоже, ко дну идти не собирается. Что помогает ей так долго жить? Какие испытания ждут ее впереди?

Расс: Система живет, потому что ее поддерживает не несколько человек, она изменила всю культуру компании. Спасибо группе первых энтузиастов, «туалетному

тестированию», рассылкам, техническим докладам, изменениям в гайдлайнах по написанию кода и в требованиях вакансий, — теперь от всех инженеров компании ждут регулярного тестирования. Участвует команда в тест-сертификации или нет, она должна хорошо продумать стратегию автоматизированного тестирования — либо самостоятельно, либо с помощью экспертов.

Программа продолжает жить, потому что доказала свою эффективность. У нас осталось очень мало областей, в которых хотя бы малая часть тестирования выполняется вручную или передается подрядчикам. В этой битве тест-сертификация победила. Даже если программа когда-нибудь завершится, этот вклад не будет забыт.

— Дайте советы инженерам из других компаний, которые собираются запустить подобные программы у себя.

Трэйси: Начинайте с команд, которые уже хорошо настроены по отношению к тестированию. Вырастите ядро из команд, которым ваша программа быстро принесет практическую пользу. Не стесняйтесь пропагандировать программу и просить об этом других. Наставничество — важный элемент успеха программы тест-сертификации. Когда вы предлагаете команде попробовать что-то новое или усовершенствовать старое, дело пойдет быстрее, если вы выделите им человека, к которому можно обращаться за помощью. Инженеру или команде бывает неудобно задавать кажущиеся глупыми вопросы в общей рассылке, но они охотно обратятся с теми же вопросами к ментору.

Постарайтесь сделать процесс веселым. Попробуйте придумать более удачное название без слова «сертификация», чтобы не кормить бюрократов. Или сделайте как мы — используйте это слово как ложную мишень и постоянно напоминайте своей аудитории, что это неудачное название, ведь ваша программа как раз «не из таких». Опишите небольшие шаги, чтобы команды могли быстро увидеть и показать другим свой прогресс. Не пытайтесь создать идеальную систему с идеальными показателями. Идеала для всех не существует. Очень важно принять приемлемое решение и двигаться вперед, не зависая на попытках достижения несбыточного идеала. Будьте гибкими там, где требуется, но стойте на своем в принципиальных вопросах.

На этом глава, посвященная жизни разработчика в тестировании, подходит к концу. В оставшейся части мы собрали дополнительный материал о том, как Google проводит собеседования с разработчиками в тестировании, и интервью с Тедом Мао, который рассказывает о некоторых инструментах разработчиков в тестировании.

Как мы собеседуем на позицию разработчиков в тестировании

Хорошие разработчики в тестировании сильны во всем — они и достаточно сильные программисты, чтобы писать код, и достаточно опытные тестировщики, чтобы протестировать что угодно. Они могут сами организовать свою работу и подобрать инструменты. Крутой разработчик в тестировании может увидеть весь лес целиком и разглядеть в нем отдельное дерево. Ему достаточно взглянуть на прототип функции или API, и он представит, как этот код будет использоваться и что в нем может сломаться.

Весь код в Google находится в общем хранилище. Это значит, что любой код может использовать кто угодно и когда угодно. Значит, код должен быть надежным.

Разработчики в тестировании не только отлавливают баги, пропущенные программистами. Они создают инструменты и следят, чтобы использование кода или компонента было очевидным для других инженеров, и постоянно думают о том, что будет с этим кодом дальше. Google двигается быстро, поэтому код должен всегда оставаться чистым, последовательным и работать, даже если его создатель давно про него забыл.

Как мы строим собеседование, чтобы найти нужные навыки и особый тип мышления? Это непросто. Но мы уже нашли сотни подходящих инженеров. Мы ищем гибрид: разработчика с сильным интересом и способностями к тестированию. Простой и эффективный способ выявить перспективных разработчиков в тестировании — давать им те же задачи, что и разработчикам, и смотреть, как они подходят к обеспечению качества и тестированию. Получается, у разработчика в тестировании вдвое больше шансов провалить собеседование.

Самые простые вопросы лучше всего помогают понять, получится ли из кандидата хороший разработчик в тестировании. Мы не тратим время на хитроумные задачи программирования и не спорим об академической правильности — это потеря времени, которое можно потратить, чтобы проверить, как кандидат подходит к программированию и качеству. Всегда будут люди, которые нарушают привычный алгоритм решения задачи. И тут важно обратить внимание на то, как кандидат *обдумывает* решение, а не на изящность самого решения.

НА ЗАМЕТКУ

Собеседуя разработчиков в тестировании, обращайте внимание на то, как кандидат обдумывает решение, а не на изящность самого ответа.

Вот пример простой задачи для разработчиков в тестировании. Представьте, что сегодня ваш первый рабочий день и вас попросили реализовать функцию `aconut(void* s)`, которая возвращает число букв «A» в строке.

Кандидат, который с ходу бросается писать код, как будто говорит нам: здесь можно сделать только одно, и я это уже делаю. Все, на что он способен, — писать код. Разработчик в тестировании не должен так узко видеть мир. Мы хотим слышать его вопросы: для чего будет использоваться эта функция? зачем мы ее создаем? похож ли прототип функции на правду? Мы хотим видеть, что кандидата волнует правильность решения и проверка поведения. Задача заслуживает, чтобы ее решали с большим уважением.

Кандидат, который без особых раздумий бросается на программистскую задачу, сделает то же самое с задачей на тестирование. Если мы просим добавить тестовые варианты в модули, то кандидат не должен очертя голову выдавать все подряд тесты, он должен начинать с лучших.

У разработчика в тестировании обычно ограничено время. Мы хотим, чтобы кандидат остановился и нашел самый эффективный способ решения задачи или улучшил существующее решение. Хороший разработчик в тестировании найдет плохо определенные функции API и по ходу тестирования превратит их в нечто красивое.

Достойный кандидат потратит несколько минут на анализ спецификации, задавая вопросы и записывая утверждения.

- Какая кодировка используется во входной строке: ASCII, UTF-8 или что-то еще?
- Имя функции слабое. Не стоит ли использовать стиль CamelCase¹ и сделать его более содержательным? Или здесь действуют другие стандарты выбора имен?
- Какой тип должна возвращать функция? Наверное, интервьюер забыл, поэтому я добавлю `int` в начало прототипа функции.
- Конструкция `void*` опасна. Лучше использовать подходящий тип, например `char*`, чтобы не пострадать от проверки типов при компиляции.
- Что понимается под буквой «А»? Нижний регистр тоже считается?
- Нет ли такой функции в стандартной библиотеке? (Тут мы ответим, что в интересах собеседования ему нужно представить, что он первым реализует эту функцию.)

Лучшие кандидаты пойдут еще дальше.

- **Подумают о масштабе.** Возможно, для возвращаемого значения лучше использовать `int64`, потому что Google часто имеет дело с большими объемами данных.

¹ CamelCase — стиль написания составных слов, при котором несколько слов пишутся слитно без пробелов, при этом каждое слово пишется с заглавной буквы. Стиль получил свое название из-за того, что заглавные буквы внутри слова напоминают горбы верблюда.

- **Подумают о повторном использовании:** почему эта функция подсчитывает только буквы «А»? Вероятно, ее стоит параметризовать, чтобы подсчитывать произвольные символы. Это лучше, чем определять разные функции для разных символов.
- **Подумают о безопасности:** эти указатели получены из надежного источника?

Наконец, самые лучшие кандидаты:

- **Подумают о масштабе:**
 - Будет ли эта функция выполняться как часть вычислений MapReduce¹ для сегментированных² данных? Возможно, это самая частая форма вызова такой функции. Есть ли проблемы, которые стоит учитывать в этом сценарии? Надо продумать правильность и быстродействие этой функции при ее выполнении для каждой страницы в интернете.
 - Если функция вызывается для каждого запроса Google и только с безопасными указателями, потому что указатели уже проверяются уровнем выше, возможно, следует перестать каждый раз проверять на `null`. Это сэкономит нам сотни миллионов циклов работы процессора и немного уменьшит задержку на пользовательской стороне. Как минимум, нужно вспомнить о возможных последствиях полной проверки параметров.
- **Подумают об оптимизациях, основанных на инвариантах:**
 - Можно ли предполагать, что входные данные уже отсортированы? Если так, то функция может прекратить свою работу, обнаружив первую букву «В».
 - Какую структуру имеют входные данные? Что чаще приходит: только одни буквы «А», сочетание любых символов или только «А» с пробелами? В зависимости от структуры можно будет оптимизировать операции сравнения. При работе с большими данными даже мелкие изменения могут сильно повлиять на задержки при выполнении кода.

1 MapReduce – технология распределенных вычислений, при которой вычислительная задача разбивается на небольшие части, которые затем собираются вместе. См. <http://en.wikipedia.org/wiki/MapReduce>.

2 Сегментация (sharding) – разновидность распределения базы данных. Горизонтальная сегментация позволяет проектировать базы данных: строки таблицы базы данных хранятся по отдельности (в отличие от вертикального разбиения по столбцам). См. [http://en.wikipedia.org/wiki/Shard_\(database_architecture\)](http://en.wikipedia.org/wiki/Shard_(database_architecture))

— **Подумают о безопасности:**

- Если эта функция является частью кода, чувствительного к безопасности, то, может быть, стоит проверить не только ненулевые указатели в общем виде. В некоторых системах единица тоже является недействительным значением для указателя.
- Если учитывать параметр длины строки, можно убедиться, что код не выходит за конец строки. Хорошо бы проверить значение параметра длины для надежности. Строки, завершающиеся null-символом, — лучшие друзья хакера.
- Если существует вероятность того, что буфер может быть модифицирован другим потоком во время выполнения этой функции, это может создать проблемы с безопасностью.
- Должна ли эта проверка выполняться в конструкции `try-catch`? Если вызывающий код не ожидает исключений, вероятно, ему следует вернуть код ошибки. Если существуют коды ошибок, насколько четко они определены и задокументированы?

Так кандидат показывает, что он думает широко, учитывая окружение кода своей функции и условия ее выполнения. Такой подход помогает избежать путаницы и упущений в дальнейшем.

В итоге самые лучшие кандидаты найдут свой угол зрения на эти вопросы. Любой способ решения задачи хорош, если он обоснован.

НА ЗАМЕТКУ

Хорошему кандидату на роль разработчика в тестировании не нужно напоминать, что написанный им код нужно тестировать. Он должен считать тестирование частью решения.

На самом деле любой инженер, прошедший вводный курс программирования, сможет написать код для решения этой задачи. А вот вопросы и подход к решению отличают лучших кандидатов. Мы стараемся создать комфортную обстановку на собеседовании, чтобы кандидат не стеснялся задавать вопросы. Если он молчит, то мы можем легонько подтолкнуть его к разговору, чтобы понять, не вызвано ли стремление поскорее погрузиться в код атмосферой интервью. Сотрудники Google должны уметь задавать любые вопросы, лишь бы это помогло решению задачи.

Было бы скучно перебирать сотни правильных реализаций и распространенные ошибки — все-таки книга не о программировании и не о проведении собеседований. Поэтому мы покажем только одну простую и очевидную реализацию. Заметим, что кандидаты обычно используют язык, который знают лучше всего, например Java или Python. При этом стоит задать дополнительные вопросы, чтобы

убедиться, что они правильно понимают специфические детали языка: как язык работает с памятью и типами данных, особенности компиляции и выполнения.

```
int64 Account(const char* s) {
    if (!s) return 0;
    int64 count = 0;
    while (*s++) {
        if (*s == 'a') count++;
    }
    return count;
}
```

Кандидат должен уметь объяснять свой код, показывать, как меняются значения указателей и счетчиков в процессе выполнения кода с тестовыми входными значениями.

Еще раз. Достойный кандидат на роль разработчика в тестировании:

- Решает эту задачу без особых проблем. Он пишет код с первого раза, не путается в базовом синтаксисе и не смешивает конструкции из разных языков.
- Правильно понимает работу указателей и не засоряет память.
- Реализует проверку входных данных, чтобы избежать проблем, связанных с `null`-указателями, или может объяснить, почему он этого не делает.
- Понимает, что время выполнения его кода линейно зависит от объема входных данных. Если зависимость нелинейная, это может говорить о творческом подходе, но может оказаться просто ошибкой.
- Исправляет мелкие недочеты в коде, если вы на них укажете.
- Пишет четкий и легко читаемый код. Если кандидат использует побитовые операции или пишет несколько команд в одной строке — это плохой признак, даже если код работает.
- Может рассказать последовательность выполнения своего кода для тестового ввода «A» или `null`.

Более талантливые кандидаты способны на большее. Они:

- Предлагают использовать `int64` для типа счетчиков и возвращаемого значения, чтобы обеспечить будущую совместимость и предотвратить переполнение, если кто-то использует функцию для подсчета букв «A» в очень длинной строке.
- Пишут код, который работает для сегментированного или распределенного выполнения вычислений. Некоторые кандидаты, не знакомые с MapReduce, могут самостоятельно прийти к простым способам уменьшить задержки, обрабатывая большие строки параллельно.

- Записывают свои предположения в примечаниях или комментариях к коду.
- Используют много вариантов входных данных и исправляют все найденные баги. Если кандидат на роль разработчика в тестировании не ищет и не исправляет баги — это тревожный признак.
- Тестируют свои функции до того, как их попросят. Наши люди должны тестировать без напоминаний.
- Продолжают попытки оптимизировать свое решение до тех пор, пока их не попросят остановиться. Никто не может быть уверен, что его код идеален, через пару минут программирования и прогона нескольких тестовых вариантов входных данных. Кандидаты должны стремиться к тому, чтобы в итоге все работало правильно.

Дальше мы должны понять, сможет ли наш кандидат протестировать свой код. Запутанный или нелогичный тестовый код — это, наверное, самое худшее в мире. Уж лучше вообще без тестов. При отладке упавшего теста должно быть очевидно, что этот тест делал. Если это не так, разработчики могут отключить тест, пометить его как ненадежный или проигнорировать сбой — такое бывает. В том, что плохой тестовый код оказался в репозитории, виноваты все, кто писал и рецензировал код.

Разработчик в тестировании должен уметь тестировать и методом черного ящика, предполагая, что функцию написал кто-то другой, и методом белого ящика, зная, какие тест-кейсы не имеют смысла из-за особенностей реализации.

Подытожим. Признаки достойного кандидата:

- Действует методично и систематично. Выстраивает тестовые данные по определенной понятной схеме, например по размеру строки, а не просто выдает в виде случайного набора.
- Фокусируется на генерации реально полезных тестовых данных. Думает о том, как проводить большие тесты и где взять реальные тестовые данные.

Признаки лучшего кандидата:

- Пытается реализовать выполнение функции в параллельных потоках, чтобы выявить перекрестные помехи, дедлоки и утечку памяти.
- Создает тесты с большой продолжительностью выполнения. Например, запускает тесты в цикле `while(true)`, чтобы убедиться, что они не падают со временем.
- Не перестает выдавать тестовые сценарии и предлагать новые подходы к тестированию, выбору данных, проверке и выполнению тестов.

Пример отличного кандидата

Джейсон Арбон

Одного нашего кандидата (который, кстати, уже великолепно справляется с работой в Google) спросили, как бы он организовал тестирование граничных условий для версии этой функции с 64-разрядными целыми числами. Он быстро догадался, что задача физически неразрешима из-за ограничений по времени и объемам данных. Но из любопытства, подогретого нетривиальностью задачи, прикинул, как можно хотя бы разместить большие объемы данных для таких тестов. В качестве входных данных он взял веб-индекс Google.

Как же он проверил свое решение? Кандидат предложил использовать параллельную реализацию и посмотреть, дадут ли обе реализации одинаковый результат. Он предложил применить метод статического выборочного контроля: знаем ли мы, какова предполагаемая частота появления буквы «А» на веб-страницах? Если нам известно количество проиндексированных страниц, мы можем сравнить результат вычислений программы с результатом, полученным теоретическим путем. Это подход к тестированию в духе Google. Хотя мы не стали реализовывать эти гигантские тесты, сама возможность таких решений наталкивает на интересные идеи для реальной работы.

Еще один параметр, который мы проверяем на собеседовании, — «гугловость», то есть соответствие нашей культуре. Насколько кандидат технически любознателен? Может ли он интегрировать в свое решение новые идеи? Как он справляется с неоднозначностью? Знакомы ли ему академические методы проверки качества, например доказательства теорем? Разбирается ли он в метриках качества или автоматизации в других областях, например в самолетостроении?

Пытается ли он оправдать наличие багов, которые мы нашли в его реализации? Может ли мыслить масштабно? Кандидаты, конечно, не обязаны обладать всеми перечисленными свойствами, но чем больше совпадений, тем лучше. И наконец, хотим ли мы вообще работать с этим человеком каждый день?

Важное замечание: если человек, проходящий интервью на должность разработчика в тестировании, оказался не таким уж сильным программистом, из него может получиться хороший инженер по тестированию. Некоторые из наших лучших инженеров по тестированию изначально пробовались на должность разработчика в тестировании.

Интересно, что мы часто упускаем хороших кандидатов, потому что за время собеседования они склоняются в сторону чистого программирования или, наоборот, зацикливаются на тестировании. Мы хотим, чтобы кандидатов собеседовали специалисты разных должностей, которым потом придется с ними работать, потому что роль разработчика в тестировании — это сплав двух ролей, и иногда ее

сложно правильно оценить на собеседовании. Нужно быть уверенным в том, что плохие оценки выставлены людьми, понимающими такую двуликийность хорошего кандидата.

Пат Коупленд во введении говорит, что по поводу найма разработчика в тестировании было и есть много полярных мнений. Если он хорошо программирует, может быть, поручать ему писать фичи? Ведь хороших разработчиков тоже сложно найти. А может, им стоит сосредоточиться только на тестировании, если они легко с этим справляются? Истина, как обычно, лежит где-то посередине.

Поиск хорошего кандидата на роль разработчика в тестировании — дело хлопотное, но оно того стоит. Даже один хороший разработчик в тестировании может оказать огромное влияние на работу всей команды.

Интервью с разработчиком инструментов Тедом Мао

Тед Мао — разработчик Google, который занимается исключительно инструментами тестирования. Он создает инструменты тестирования веб-приложений, которые масштабируются на все, что создается в Google. Тед хорошо известен среди разработчиков в тестировании, чья работа немыслима без хороших инструментов. Тед, вероятно, знает общую инфраструктуру веб-тестирования Google лучше всех.

— Когда ты пришел в Google и чем тебя привлекла эта работа?

Тед: Я присоединился к Google в июне 2004 года. До этого я работал только в крупных компаниях, таких как IBM и Microsoft, а тут появился Google — перспективный стартап, который привлекал многих талантливых инженеров. Казалось, что здесь будет много интересных и сложных задач, и я хотел поучаствовать в этой работе рука об руку с лучшими инженерами в мире.

— Ты придумал и реализовал Buganizer¹, багтрекинговую систему Google. Чего ты хотел добиться в первую очередь с помощью Buganizer и чем эта система была лучше старой BugDB?

Тед: Система BugsDB не только не поддерживала, а даже затрудняла наш процесс разработки. Честно говоря, на работу с ней уходило слишком много времени. Это была своего рода дань, которую она взимала с каждой команды. Она создавала проблемы по всем фронтам: пользовательский интерфейс тормозил, жизненный цикл бага был устроен ужасно, текстовые поля были плохо структурированы и требовали вводить данные особым образом. Проектируя Buganizer, мы позаботились о том, чтобы наша модель данных и пользовательский интерфейс отвечали реальным

¹ Версия Buganizer с открытым кодом, которая называется Issue Tracker, доступна в проекте Chromium по адресу <http://code.google.com/chromium/issues/list>

процессам разработки наших внутренних пользователей. Мы построили систему, которую можно было расширять в дальнейшем как саму по себе, так и с помощью интеграции.

— Итак, ты создал Buganizer. Правда, это лучшая багтрекинговая система, из всех, с которыми нам доводилось работать. А как ты начал заниматься автоматизацией веб-тестирования? Ты понял, что есть потребность, или тебя попросили решить проблему с выполнением тестов?

Тед: Занимаясь Buganizer, AdWords и другими продуктами Google, я постоянно наталкивался на то, что существующей инфраструктуры веб-тестирования не хватает для моей работы. Она никогда не была настолько быстрой, масштабируемой, надежной и полезной, насколько мне было нужно. Когда инструментальная команда объявила поиск рулевого для нового направления, я вызвался. Проект получил название Matrix, и я его возглавил.

— Сколько тестовых прогонов и команд поддерживает Matrix сегодня?

Тед: Зависит от того, как измерять количество прогонов и команд. Например, одна из наших метрик называется «сессия браузера». Все новые сессии одного конкретного браузера по умолчанию начинаются в одном состоянии, а значит, выполнение теста в браузере будет детерминировано так же, как сам тест, браузер и ОС. Matrix поддерживает больше миллиона сессий браузера каждый день и используется почти всеми фронтенд-командами в Google.

— Сколько людей работало над этими двумя проектами: Buganizer и Matrix?

Тед: В самые жаркие периоды разработки над Buganizer работало пять инженеров, а над Matrix четыре. Я всегда расстраиваюсь, думая о том, чего бы мы добились с постоянной большой командой. Но, думаю, мы все-таки прекрасно справились теми силами, которыми располагали.

— Какие самые серьезные технические проблемы вы преодолевали, разрабатывая инструменты?

Тед: Самые сложные и зачастую самые интересные испытания всегда приходятся на этап проектирования: разобраться в проблеме, сравнить разные решения, а потом сделать правильный выбор. После вашего выбора реализация уже пойдет по накатанному пути. Эти решения принимаются один раз и на всю жизнь проекта. Вместе с реализацией они определяют успех или неудачу продукта.

— Что ты посоветуешь программистам, которые разрабатывают средства тестирования?

Тед: Сосредоточьтесь на своих пользователях, поймите, что им нужно, и решите их проблемы. Не забывайте о невидимых фичах вроде удобства использования

и скорости. Инженеры умеют решать свои проблемы как никто другой, так дайте им возможность использовать ваши инструменты так, как им удобно.

— Какой ты видишь следующую масштабную или просто интересную проблему, которую нужно решить в области тестовых инструментов и инфраструктуры?

Тед: В последнее время я все чаще думаю о том, что наши инструменты становятся более мощными и сложными. И следовательно, все более трудными для понимания и использования. Например, в нашей текущей инфраструктуре веб-тестирования инженер может одной командой запустить параллельное выполнение тысяч веб-тестов в разных браузерах. С одной стороны, очень хорошо, что мы не вязнем в деталях происходящего: где именно выполняются эти тесты, откуда берутся браузеры, как определяется конфигурация среды тестирования и т. д. С другой стороны, если тест не пройдет и инженер займется его отладкой, все эти подробности все равно понадобятся. У нас уже есть несколько наработок в этой области, однако можно и нужно сделать гораздо больше.

Интервью с создателем WebDriver Саймоном Стюартом

Саймон Стюарт — создатель WebDriver и гуру браузерной автоматизации в Google. WebDriver — это опенсорс-инструмент для тестирования веб-приложений, который популярен как в Google, так и за его пределами. Исторически WebDriver — это одна из самых горячих тем на конференции Google Test Automation Conference.

Авторы встретились с Саймоном, чтобы узнать, что он думает об автоматизации тестирования веб-приложений и о будущем WebDriver.

— Мне кажется, многие люди не совсем понимают, в чем отличие Selenium от WebDriver. Ты можешь объяснить разницу?

Саймон: Джейсон Хаггинс начал проект Selenium, когда работал в ThoughtWorks¹. Джейсон писал веб-приложение и ориентировался преимущественно на браузер IE, который занимал тогда больше 90% рынка. Но он постоянно получал сообщения о багах от пользователей, пересевших на Firefox, причем исправление ошибки для Firefox могло нарушить работу приложения в IE. Джейсон придумал Selenium как механизм, запускающий и тестирующий приложение в обоих браузерах.

Я начал разрабатывать WebDriver примерно через год после создания Selenium, но еще до того, как проект Джейсона вышел на стабильность. Я ориентировался

¹ ThoughtWorks, Inc — IT-компания, основанная в 1983 году в Чикаго. Сейчас ее офисы открыты в 11 странах мира. Компания считается мировым лидером по разработке ПО для гибкой разработки. — Примеч. перев.

на более общее тестирование веб-приложений, поэтому неудивительно, что мы выбрали разные способы реализации. Проект Selenium строился на запуске JavaScript-кода в браузере, а WebDriver сам интегрировался в браузер через API. У каждого подхода были свои достоинства и недостатки. Selenium почти сразу же мог работать с новыми браузерами, например с Chrome, но не умел загружать файлы и нормально обрабатывать действия пользователя. Все-таки возможности песочницы JS ограничены. Механизм WebDriver был встроен в браузер, поэтому он мог обходить эти ограничения, но добавлять другие браузеры было тяжело. Когда мы оба пришли в Google, мы решили объединить проекты.

— Но мы до сих пор слышим оба названия. Это все еще разные проекты?

Саймон: Selenium — это «зонтичный» проект, под которым мы выпускаем все инструменты браузерной автоматизации, среди которых есть и WebDriver. Официально он называется Selenium WebDriver.

— А как Google начал участвовать в этом?

Саймон: Когда открылся лондонский офис Google, туда пригласили работать нескольких бывших сотрудников ThoughtWorks. Они пригласили меня рассказать о WebDriver. Тот доклад у меня не удался — какой-то парень в первом ряду все время засыпал, и мне пришлось перекрикивать его храп в аудитории. Для полного счастья что-то произошло с аппаратурой, и мой доклад не записался. Несмотря на все злоключения, проектом заинтересовались, и меня пригласили провести презентацию уже без храта на конференции GTAC. Вскоре я начал работать в Google. И теперь знаю все их страшные тайны.

— Ну конечно, все наши шкафы забиты скелетами. Вообще-то мы видели, как ты выступаешь, и не можем представить, чтобы кто-то заснул на твоем докладе. Мы его знаем?

Саймон: Нет. Он уже давно покинул Google. Будем считать, что у него выдалась тяжелая ночь.

— Пусть это будет уроком нашим читателям. Засыпать на докладе Саймона Стюарта вредно для вашей карьеры. WebDriver стал твоей основной работой, когда ты присоединился к Google?

Саймон: Нет, это был мой «двадцатипроцентный» проект. Хотя в основное время я был разработчиком в тестировании, мне удавалось двигать WebDriver вперед. Очень помогало участие разработчиков со стороны. На ранней стадии опенсорс-проекта людям приходится вкладываться в проект, потому что это единственный вариант его развития. Теперь многие пользователи WebDriver узнают о проекте от других и относятся к нему скорее как потребители, чем как участники. Но в те дни WebDriver развивался силами энтузиастов сообщества.

— Что ж, мы знаем, к чему это привело — WebDriver стал очень популярен в Google. С чего все началось? С одного пилотного проекта? Были фальстарты?

Саймон: Все началось с Wave — социального продукта, который был создан в нашем сиднейском филиале. С тех пор проект успели закрыть. Разработчики Wave попытались использовать Selenium в качестве тестовой инфраструктуры, но у них это не вышло. Wave оказался слишком сложным продуктом. Разработчики были достаточно упрямые, чтобы докопаться до WebDriver. Они начали задавать мне много вопросов. Хороших вопросов.

Работы стало больше, чем я мог выполнять в свои «20 процентов». Они переговорили с моим руководством и договорились о месячной командировке. Так я отправился в Сидней, чтобы помочь им построить тестовую инфраструктуру.

— Насколько я понимаю, поездка увенчалась успехом.

Саймон: Да, команда очень сильно помогла, и мы справились. Та работа поставила новые требования перед моим продуктом и показала другим командам Google, что WebDriver — подходящий инструмент для современных веб-приложений. С этого момента у WebDriver не было недостатка в пользователях, а у меня появилась возможность работать над ним в основное время.

— С первым пользователем всегда сложнее всего. Как ты справился с необходимостью разрабатывать WebDriver и одновременно адаптировать его для Wave?

Саймон: Я пользовался процессом DDD (Defect-Driven Development — разработка через дефекты). Я объявлял WebDriver бездефектным, а когда пользователь находил баг, я его исправлял и снова объявлял продукт бездефектным. Так я исправлял только по-настоящему значимые для людей баги. Этот процесс идеален для доработки существующего продукта. Вы исправляете только важные баги, а не возитесь с дефектами, до которых никому нет дела.

— Ты так и остаешься единственным разработчиком WebDriver?

Саймон: Нет, теперь у нас сформировалась команда. Мы — официальный внутренний проект Google и пользуемся активной поддержкой в опенсорс-сообществе. Число браузеров, версий и платформ постоянно растет. Я говорю людям, что мы тут все сумасшедшие и все же каждый день делаем невозможное. Думаю, большинство разработчиков в здравом уме от такого будут держаться подальше.

— Итак, после Wave вы набрали скорость. Именно в этот момент WebDriver стал отделяться от старой инфраструктуры Selenium? Мы имеем в виду в контексте пользователей.

Саймон: Думаю, да. Многие разработчики Selenium переключились на другие проекты, а у меня успех WebDriver с Wave вызвал приток энергии. Люди, которых

я не знал лично, например Майкл Там из Германии, начали делать важную работу для WebDriver, и я старался бережно поддерживать эти отношения. Майкл стал первым человеком, не знакомым мне лично, который получил право заливать код в репозиторий проекта.

Но я не так уж пристально следил за распространением WebDriver. Понятно, что чем ближе команда находится ко мне физически, тем больше вероятность их перехода на WebDriver. Думаю, моими первыми пользователями еще до Wave стала команда Picasa Web Albums, а потом WebDriver подхватила команда Ads. По части использования веб-автоматизации в Google нет единства. Chrome использует Ru-Auto, Search использует Puppet (у которого есть версия с открытым кодом — Web Puppeteer), Ads использует WebDriver и т. д.

— Как насчет будущего WebDriver? К чему стремится ваша команда?

Саймон: Сейчас на поле вышло много игроков. Всего пару лет назад у нас был рынок с одним доминирующим браузером. Это в прошлом. Internet Explorer, Firefox, Chrome, Safari, Opera — далеко не все примеры, и это только десктоп. Браузеры на базе WebKit для мобильных устройств плодятся бешеными темпами. Коммерческие инструменты игнорируют все, кроме IE. Это кажется безумием, ведь 2008 год давно прошел. Следующим логичным шагом станет стандартизация WebDriver, чтобы мы могли гарантировать работу веб-приложений в разных браузерах. Конечно, было бы полезно сотрудничать с разработчиками браузеров, чтобы мы могли обеспечивать совместимость с WebDriver API.

— Похоже, это задача для комитета по стандартизации. Есть прогресс в этом направлении?

Саймон: Да. К сожалению, это означает, что мне приходится писать английские буквы вместо кода, зато у комитета W3C¹ появилась наша спецификация, ведь разработчики браузеров связаны с этой организацией.

— Чего ты ожидаешь в будущем? Как средства автоматизации браузеров будут развиваться дальше?

Саймон: Надеюсь, они отойдут на второй план. Рассчитываю, что любой браузер будет поддерживать автоматизацию API и люди наконец перестанут думать об инфраструктуре, а просто будут использовать ее. Я хочу, чтобы разработчики концентрировались на новых фичах своих веб-приложений, а не на том, как их автоматизировать. WebDriver станет по-настоящему успешным, когда разработчики перестанут замечать, что он вообще используется.

¹ World Wide Web Consortium (сокращенно W3C) — Консорциум Всемирной паутины, который разрабатывает и внедряет технологические стандарты для всего интернета. — Примеч. перев.

ГЛАВА 3

Кто такой инженер по тестированию

Разработчики в тестировании в Google создают, настраивают, поддерживают и развиваются инфраструктуру для автоматического тестирования. Инженеры по тестированию играют другую роль — они ориентированы на «человеческую сторону»: как пользователи будут взаимодействовать с приложением и какие проблемы при этом могут возникнуть. Им, как и большинству технических специалистов в Google, нужно уметь программировать. Но все-таки для решения многих задач тестировщикам не нужно писать код¹.

Тестирование, обращенное к пользователю

В предыдущей главе мы описали инженера по тестированию как своеобразного «разработчика во имя пользователя», и это действительно так.

В Google есть особое, ставшее уже частью культуры компании, почтение к программированию. Принято считать, что любой, кто входит в команду разработки,

¹ Это довольно общее описание. На самом деле многие тестировщики в Google делают почти то же, что и разработчики в тестировании, и тогда они пишут много кода. Других можно приравнять к релиз-менеджерам, они пишут очень мало кода.

должен знать, как писать код. Поэтому тестировщики должны уметь программировать, если хотят работать наравне с разработчиками. Тестировщик в Google, с одной стороны, обладает техническими навыками, которые так уважают разработчики, а с другой — умеет фокусироваться на потребностях пользователей, что постоянно держит программистов в тонусе. Вот такое вот раздвоение личности!

НА ЗАМЕТКУ

Тестировщики должны уметь программировать, если хотят работать наравне с разработчиками. Тестировщик в Google, с одной стороны, обладает техническими навыками, которые так уважают разработчики, а с другой — умеет фокусироваться на потребностях пользователей, что постоянно держит программистов в тонусе.

Должностные обязанности тестировщиков сложно описать однозначно. Они отслеживают все, что связано с качеством, на протяжении всего процесса разработки, по мере того как одиночные сборки постепенно складываются в один большой продукт. Поэтому многие из них вовлекаются в самую низкоуровневую работу, где требуется еще одна пара опытных глаз и инженерный опыт. Это вопрос рисков: тестировщики находят самые уязвимые места в продукте и делают то, что принесет максимум ценности конкретно в этом случае. Иными словами, тестировщик будет выполнять задачи разработчика в тестировании, если именно эта работа принесет сейчас наибольшую пользу. Если нужно провести, например, код-ревью, он сделает это. Если тестовая инфраструктура хромает — тестировщик займется ею. Один и тот же тестировщик может сегодня заниматься исследовательским тестированием, а завтра в тестовых целях сам будет использовать продукт как обычный пользователь.

Иногда распределение работы зависит от времени: на ранних стадиях проекта надо решать задачи в роли разработчика в тестировании, а позже возникает потребность поработать именно тестировщиком. В других случаях тестировщик сам выбирает, какую работу ему выполнять здесь и сейчас. Здесь нет четких правил. То, что мы опишем в следующем разделе, по сути, является идеальным случаем.

Инженер по тестированию

Инженеры по тестированию появились в Google позже разработчиков и даже позже разработчиков в тестировании, поэтому эта роль до сих пор находится в процессе становления. Сегодняшние тестировщики определяют ориентиры, которыми будут руководствоваться следующие поколения. В этой части мы расскажем о новейших процессах, которые используют такие инженеры в Google.

Не все проекты требуют внимания инженеров по тестированию. Некоторые проекты находятся в начальной стадии и не имеют четкой задачи или описанных пользовательских сценариев. Таким проектам вряд ли будет уделяться особое (да и любое другое) внимание тестировщиков. Если есть большая вероятность закрытия проекта (например, он создавался для проверки гипотезы и не выдержал испытания), если в нем еще нет пользователей или четкого набора фич, то тестирование должны выполнять сами разработчики.

Даже если продукт точно будет выпущен, то работы для тестировщика на ранней стадии разработки не много, так как функциональность еще не устоялась, не определены основные фичи и границы проекта. Если разработчик в тестировании уже задействован, привлечение инженера по тестированию может стать лишней тратой сил и времени. К тому же в самом начале для планирования тестирования нужно намного меньше тестировщиков, чем для исследовательского тестирования на финальных циклах, когда разработка продукта близка к завершению и охота за пропущенными дефектами становится самой важной задачей.

НА ЗАМЕТКУ

Работы для тестировщика на ранней стадии разработки немного, так как функциональность еще не устоялась, не определены основные фичи и границы проекта.

Хитрость с подключением тестировщиков к проекту в том, что нужно учитывать риски и степень выгоды от работы тестировщиков. Если потенциальный риск для будущего пользователя (или для всей компании) считается высоким, тогда тестировщиков нужно больше, но затраченные усилия должны быть соизмеримы с возможной выгодой от проекта. Если вы решили подключать тестировщиков, то обязательно соблюдайте правильные пропорции и делайте это в подходящий момент.

К моменту, когда в дело вступают тестировщики, разработчики и разработчики в тестировании уже выполнили большой фронт работ, связанных с организацией тестирования и обеспечением качества. Поэтому тестировщикам не нужно начинать все с нуля. Их работа в проекте начинается с ответов на вопросы:

- Где у продукта слабые места?
- Есть ли проблемы с безопасностью, конфиденциальностью, производительностью, надежностью, пригодностью использования, совместимостью?
- Работают ли первичные пользовательские сценарии как следует? Подходят ли они для международной аудитории?
- Взаимодействует ли продукт с другими (аппаратными и программными) продуктами?
- Хорошо ли работают средства диагностики проблем?

На самом деле вопросов, на которые предстоит ответить тестировщикам, гораздо больше, и все они связаны с выявлением рисков при выпуске продукта. И совсем не обязательно тестировщики выполняют всю работу сами, их задача в том, чтобы работа была выполнена. Для этого они могут привлекать других людей там, где это нужно. Так или иначе, тестировщикам платят за то, чтобы они защищали пользователей и бизнес от плохого проектирования, запутанного интерфейса, багов, проблем с безопасностью, конфиденциальностью и т. д. В Google только эти специалисты тратят все свое рабочее время на то, чтобы искать слабые места в приложениях. Поэтому эта роль намного меньше формализована, чем, например, роль разработчика в тестировании. К тестировщикам обращаются за помощью на всех стадиях проекта: и на этапе идеи, и во время выпуска восьмой версии, и даже когда проект уже считается древним, «законсервированным». Часто один такой инженер работает над несколькими проектами одновременно, особенно если его умения сфокусированы на безопасности, защите конфиденциальности и нюансах интернационализации.

Конечно, работа тестировщиков зависит от проекта. Одни много программируют, но держат фокус скорее на средних и больших тестах (например, сквозных пользовательских сценариях), чем на малых. Другие берут уже существующий код и ищут слабые места, которые могут привести к появлению дефектов. Занимаясь такой работой, тестировщики даже могут изменять код проекта (но не писать его с нуля!), но прежде всего они должны исполнять свою роль, то есть смотреть на всю систему целиком, фокусироваться на ее полноценном использовании. У тестировщиков есть своя суперсила: поиск неоднозначностей в требованиях и умение грамотно обосновывать свои сообщения о всевозможных существующих и вероятных проблемах.

Решая свои задачи, тестировщики постоянно сталкиваются с другими участниками команды. Найдя слабое место, наши ребята с удовольствием ломают программу и передают ее осколки в добрые руки разработчиков и руководителей проекта. Обычно тестировщики — самые известные члены команды из-за широкого круга общения, который им надо поддерживать для эффективной работы.

Такое описание функций специалистов может выглядеть пугающее. Что ж, пугаем дальше. Успешный тестировщик в Google состоит из технических умений, склонности к лидерству и глубокого понимания продукта. Действительно, без соответствующей поддержки многие могут сломаться. Поэтому в Google сформировалось сильное сообщество тестировщиков, которое помогает решить эту проблему. Пожалуй, среди тестировщиков самая крепкая поддержка внутри профессии. Чтобы быть хорошим тестировщиком, нужно быть проницательным и способным к управлению, поэтому многие топ-менеджеры Google вышли именно из тестировщиков.

НА ЗАМЕТКУ

Чтобы быть хорошим тестировщиком, нужно быть проницательным и способным к управлению, поэтому многие топ-менеджеры Google вышли именно из тестировщиков.

Любопытный момент: работа тестировщиков в Google настолько разнообразна, что формализовать процесс их участия почти невозможно. Тестировщик может присоединиться к команде разработки в любой момент, он должен быть способен оценить состояние проекта, кода, архитектуры, пользователей и оперативно решить, на чем следует сосредоточиться в первую очередь. Если проект только начинается, то надо заняться планированием тестирования. Если проект уже в разгаре, то надо оценить готовность проекта к выпуску или искать какие-то дефекты, которые еще можно поймать до выхода проекта в бета-тестирование. Если в руки тестировщика попадает продукт, который Google только что приобрел, как правило, проводится исследовательское тестирование с минимальным планированием (или вообще без него). Некоторые проекты не выпускались довольно долгое время, и с них нужно смахнуть пыль, подлатать систему безопасности или немного изменить пользовательский интерфейс — здесь нужен уже другой подход.

В Google нельзя равнять всех тестировщиков под одну гребенку. Мы часто называем таких ребят «те, кто приходит в середине проекта», а это значит, что тестировщик должен быть гибким, способным быстро интегрироваться в команду разработки и текущую ситуацию. Если уже поздно строить тест-план — к черту тест-план. Если проекту срочно нужны тесты — нужно распланировать самый минимум, чтобы можно было начать. Следует реагировать по ситуации. Не стоит зацикливаться на каких-то догмах в тестировании. Если пришел к середине проекта, то сделай эту середину золотой.

Вот общий список того, в чем тестировщик должен однозначно разбираться:

- планирование тестирования и анализ рисков;
- анализ спецификации, архитектуры, кода и существующих тестов;
- исследовательское тестирование;
- пользовательские сценарии;
- разработка тест-кейсов;
- выполнение тест-кейсов;
- организация краудсорс-тестирования¹;
- метрики использования;
- работа с обратной связью от пользователей.

¹ Краудсорсинг (англ. crowd sourcing) — привлечение к работе добровольцев из профессионального сообщества без заключения договора. — Примеч. перев.

Лучше всех эти задачи решают тестировщики с сильными лидерскими качествами и хорошими навыками общения.

Планирование тестирования

Разработчики имеют важное преимущество перед тестировщиками: ценность их работы очевидна для всех. Разработчик пишет код, код становится приложением, которого жаждет пользователь, и это приложение принесет компании прибыль. Работа разработчика по определению становится самой важной.

Тестировщики же работают с документами и артефактами, которые имеют определенный срок годности: на ранних стадиях они пишут тест-планы, позднее создают и выполняют тест-кейсы, формируют баг-репорты. Еще позже тестировщики составят отчеты о покрытии кода тестами и соберут отзывы у пользователей. Когда проект выпущен, разве кого-то интересует, как было проведено тестирование? Если продукт полюбился пользователю, то хорошее тестирование — это уже данность. Если проект потерпел неудачу, люди могут высказать свое мнение о тестировании, но вряд ли кто-то захочет подробностей.

Тестировщики не могут позволить себе эгоистично зацикливаться на тестовой документации. В муках программирования, просмотра кода, сборки, тестирования и постоянных повторениях этого цикла сложно выделить время на восхищение тест-планом. Плохому тест-кейсу редко уделяют столько внимания, чтобы его переделать, проще выкинуть и заменить рабочим. Все внимание сосредоточено на растущем коде, и это единственное, что по-настоящему важно. Так и должно быть.

Среди всех тестовых документов у тест-плана самая маленькая продолжительность жизни¹. На ранней стадии руководитель проекта часто настаивает на создании тест-плана (в приложении А приведен тест-план для ранней версии Chrome OS). Считается, что написание тест-плана — это отдельная задача со своим сроком и важностью. Но когда такой план создан, становится трудно уговорить того же руководителя проекта регулярно просматривать и обновлять его. Тест-план превращается в любимого плюшевого мишку, которого мы, как в детстве, таскаем с собой, не играя, но обязательно разревемся, если у нас его отберут.

Тест-план рождается первым, первым же он должен и умереть — отпустите его. На ранней стадии проекта тест-план еще отражает общую идею проекта, но без постоянного ухода и грамотного кормления тест-план быстро чахнет: появляется новый код, фичи изменяются и добавляются, решения, которые хорошо

¹ Конечно, в случаях когда заказчик участвует в обсуждении тестовых планов или существует некое правительственное распоряжение на этот счет, гибкость, о которой мы пишем, пропадает. Есть некоторые тестовые планы, которые нужно писать и регулярно обновлять!

смотрелись на бумаге, переосмысливаются по ходу работы и получения отзывов от пользователей.

Чтобы тест-план оставался жизнеспособным, требуется огромная бюрократическая работа, и она оправданна только в том случае, когда к этому документу регулярно обращаются заинтересованные в проекте люди.

НА ЗАМЕТКУ

Тест-план рождается первым, и первым же он должен умереть.

Этот пункт — краеугольный камень всей идеи планирования тестирования. Насколько тест-план действительно управляет тестированием на протяжении всей разработки? Обращаются ли к нему тестировщики, распределяя фичи между собой? Настаивают ли разработчики на своевременном обновлении тест-плана? Правда ли, что руководители разработки открывают тест-план на своих компьютерах так же часто, как и свои списки дел? Как часто тест-менеджер ссылается на содержимое тест-плана во время встреч, посвященных статусам и прогрессу? Если план действительно важен, то все это должно происходить каждый день.

Тест-план должен играть центральную роль во время выполнения проекта. Это должен быть документ, который родился одновременно с проектом, живет и взрослеет вместе с ним, обновляется при каждом обновлении кода проекта и описывает продукт в его текущем виде, а не в том, каким он был на старте. Тест-план должен облегчить работу инженеру, примкнувшему к проекту на любой стадии.

Но это идеальный вариант, почти сказочная ситуация, которой мало кому удавалось достичь — будь то в Google или в других компаниях.

Давайте решим, каким мы хотим видеть тест-план:

- он всегда актуален;
- он объясняет назначение продукта и то, почему пользователи его полюбят;
- он описывает структуру проекта с названиями отдельных компонентов и фич;
- он описывает, что будет делать продукт и как именно он это будет делать.

С точки зрения тестирования мы должны беспокоиться об актуальности тест-плана и в то же время не превращать его поддержание в самоцель:

- создание плана должно быть быстрым и должно давать возможность оперативного изменения;
- план должен описывать то, что надо протестировать;
- план должен помогать оценивать тестирование и выявлять пробелы в покрытии.

В Google история планирования тестирования почти такая же, как и в других компаниях. Каждая команда сама определяла то, как будет выглядеть и функционировать тест-план, исходя из принятых и удобных для нее форматов работы. Например, некоторые писали тест-планы в Google Docs в виде текстовых документов или электронных таблиц с общим доступом для своей команды. Другие хранили планы прямо на странице своего продукта. Третьи добавляли его на внутренние страницы Google Sites и включали ссылки на них в инженерную документацию и внутренние вики-системы. Были и те, кто предпочитал классику и пользовался документами Microsoft Word, которые рассыпались по почте участникам команд. У кого-то тест-планов не было вообще, а были лишь наборы тест-кейсов, которые, должно быть, и представляли такой план.

Рецензировать такие планы было сложно, очень трудно было определить авторов и рецензентов. Дата создания многих тест-планов ясно говорит, что про них давно забыли, как про просроченный джем в недрах холодильника. Когда-то они были важны, но то время давно прошло.

В Google то тут, то там возникали предложения о создании централизованной системы хранения тест-планов и шаблонов для их составления. Эта прекрасная идея, возможно, и прижилась бы в другом месте, но она явно противоречила самоуправляемой культуре Google, где концепция «большого правительства» вызывает только насмешки.

На помощь пришел ACC-анализ (Attribute Component Capability) — процесс, который сформировался из практик нескольких команд Google. Его инициаторами были авторы книги и несколько их коллег. ACC-анализ прошел фазу ранних последователей, прижился и сейчас даже экспортируется в другие компании. Его автоматизированная версия называется Google Test Analytics.

Основные принципы Attribute Component Capability:

- **Избегайте повествования, пишите списки.** Не все тестировщики хотят стать писателями, когда вырастут. Не все могут красиво описать текстом цели создания продукта или его потребности в тестировании. Прозу трудно читать, поэтому, пожалуйста, только факты!
- **Оставьте продажи в покое.** Тест-план — это не маркетинговый документ, поэтому разговоры о том, как прекрасен продукт и как он вписывается в свою рыночную нишу, здесь не нужны. Тестовые планы не для покупателей или аналитиков, они для инженеров.
- **Не лейте воду.** У тест-плана нет заранее определенного объема. Это не дипломная работа, где размер имеет значение. Больше — не означает лучше. Размер плана должен зависеть от объема тестирования, а не от склонности его автора к графомании.

- **Если какой-то пункт не важен и не требует действий, не включайте его в план.** Ни одно слово в тест-плане не должно вызывать у его читателя реакции «это для меня не важно».
- **Пишите «от общего к частному».** Каждый раздел плана должен расширять предыдущий, чтобы у читателя оставалась общая картина проекта в голове, даже если он прекратил читать. Если ему будет нужно — он продолжит чтение.
- **Направляйте процесс мышления.** Хороший процесс планирования помогает участнику тщательно продумать функциональность и потребности тестирования. Он ведет его от высокоуровневых концепций к низкоуровневым деталям, которые можно реализовать.
- **Итогом должны стать тест-кейсы.** К моменту завершения план должен не только показывать, как выполнять тестирование, но и сделать работу над тест-кейсами очевидной. Если ваш план не приводит к созданию тестов, вы потратили время зря.

НА ЗАМЕТКУ

Если ваш план не приводит к созданию тестов, вы потратили время зря.

Последний пункт — самый важный: если тест-план не описывает, какие тесты мы должны написать, то он не соответствует главной задаче, ради которой мы его писали, — помочь в тестировании. Планирование тестирования должно привести нас в точку, где мы точно знаем, какие тесты нужно написать. Еще раз: вы можете считать планирование оконченным, когда точно знаете, какие тесты нужно написать.

ACC-анализ помогает в решении этой задачи планировщику, проводя его через три представления о продукте, соответствующих:

1. *прилагательным и наречиям*, описывающим цели и назначение продукта;
2. *существительным*, определяющим различные части и фичи продукта;
3. *глаголам*, которые расскажут, что продукт будет делать.

Следуя этому плану, мы сможем протестировать, что все работает правильно, а компоненты удовлетворяют цели и назначению приложения.

A — значит Attribute

Мы начинаем планирование тестирования, или ACC-анализ, с выявления, почему этот продукт важен для пользователя и для бизнеса. Зачем мы создаем его? Какова его основная ценность? Почему он интересен пользователю? Помните, что

разработчики и руководители продукта уже сделали свою работу: они выпускают продукт, который считают востребованным, поэтому нет необходимости объяснять, почему именно он нужен на рынке. Наша задача как тестировщиков — разложить все по полочкам и наклеить ярлыки. Это поможет нам учсть нужную информацию о продукте при тестировании.

Мы ранжируем данные о продукте с помощью АСС-анализа (Attribute, Component, Capability), то есть раскладываем всю имеющуюся у нас информацию по трем полкам: атрибуты, компоненты, возможности. И, следуя принятому порядку, начинаем с определения атрибутов.

Итак, атрибуты — это «прилагательные» нашей системы. Это качества и характеристики, которые продвигают наш продукт и отличают его на рынке. Это те причины, по которым пользователи выбирают именно его. Возьмем Chrome и опишем его как «быстрый», «безопасный», «стабильный» и «стильный»: вот определения продукта, на которые мы повесим ярлык «атрибуты» и положим на соответствующую полку.

Такая маркировка здорово упростит нам работу в будущем, когда мы будем связывать тест-кейсы с этими самыми атрибутами. Мы будем четко видеть, какой объем работы по тестированию нужен, чтобы продемонстрировать, насколько Chrome быстрый, безопасный и т. д.

НА ЗАМЕТКУ

Атрибуты — это «прилагательные» нашей системы. Это качества и характеристики, которые продвигают наш продукт и отличают его на рынке. Это те причины, по которым пользователи выбирают именно его.

Менеджер продукта может помочь составить список атрибутов системы. Тестировщик составляет такой список, читая требования к продукту, изучая его концепцию, миссию или просто слушая, как специалист по продажам описывает продукт потенциальному покупателю. Мы обнаружили, что продавцы и евангелисты продукта — это превосходный источник атрибутов. Представьте рекламный текст на обратной стороне коробки или подумайте, как бы ваш продукт был представлен в интернет-магазине. Представили? Теперь у вас есть правильный ход мыслей, чтобы составить список атрибутов.

Вот несколько советов для формирования перечня атрибутов.

- **Проще.** Вы не должны тратить на составление списка больше часа или двух.
- **Точнее.** Убедитесь, что вы пользуетесь документами или данными, которые ваша команда считает правдивыми.
- **Быстрее.** Не бойтесь, если вы что-то упустите на этом этапе. Если информация важная, она всплывает в ходе работы, если же нет — беспокоиться не о чем.

- **Короче.** Не больше дюжины атрибутов — это хороший результат. Мы свели список атрибутов Chrome OS к двенадцати пунктам, но, оглядываясь назад, можем сказать, что восемь или девять было бы в самый раз.

The screenshot shows a Google Sheets spreadsheet titled "Chrome OS Risk Analysis". The sheet contains a table with columns labeled A through F. Column A is "Attribute", column B is "Component", column C is "Capability", column D is "Estimated Frequency of Failure", column E is "Estimated Impact to User", and column F is "Calculated automatically". The rows are numbered from 1 to 233. Row 1 is a header row. Rows 2 and 3 show examples of how the matrix is populated. The data includes various system components like Plugins, Power Management, and Sync, along with their associated capabilities and risk scores.

Chrome OS Risk Analysis						
	File	Edit	View	Insert	Format	Data
	Tools	Collaborate	Help	Saved seconds ago		
fx	Web Centric					
	A	B	C	D	E	F
1	Attribute	Component	Capability	Estimated Frequency of Failure	Estimated Impact to User	Calculated automatically
2	From Sheet 1	From Sheet 2	Behavior of the component in response to the feature	A choice of (very rarely, seldom, occasionally, often) {1,2,3,4}	A choice of (minimal, some, considerable, maximum) {1,2,3,4}	
204	Web Centric	Plugins	Fully supports picasa uploader	3	3	9
205	Web Centric	Plugins	Fully supports silverlight	3	2	6
206	Long Battery Life	Power Management	ARM and Intel CPU power management features	4	3	12
207	Simple, Elegant	Power Management	Responds to hardware events (lid close, power button, etc.)	3	4	12
208	Simple, Elegant	Power Management	Displays indicators for the various battery statuses	3	3	9
209	Stable	Power Management	Shutdown/Sleep the netbook gracefully on critically low battery	3	3	9
210	Long Battery Life	Power Management	power conservation modes for screens (DPMS)	3	3	9
211	Simple, Elegant	Power Management	Disable screen saver when play video	2	2	4
212	Simple, Elegant	Printing	Cloud printing support	3	3	9
213	Secure	Remote Wipe	Remote wipe of machine			
214	Web Centric	Sync	Cloud sync of network/password info	2	4	8
215	Secure	Sync	Cloud sync of network/password info			
216	Web Centric	Sync	Cloud sync of system settings	2	2	4
217	Web Centric	Sync	Cloud sync of history	2	1	2
218	Secure	Sync	Cloud sync of history			
219			Combination of both machine/user settings which are both syncable/non-syncable. Come up with most common sets of setting combos to test.			
220	Web Centric	Sync				
221	Simple, Elegant	Sync	Availability of sync service	3	4	12
222	Simple, Elegant	UI				
223	Web Centric	System Settings	All syncable control panel options synced to Gaia (will need final list from dev).			0
224	Simple, Elegant	System Settings	Network defaults & options, general configurations			0

Рис. 3.1. Анализ рисков Chrome OS

Обратите внимание, что некоторые рисунки в этой главе просто иллюстрируют написанное, не стоит в них всматриваться.

Атрибуты нужны, чтобы определить, как продукт работает на ключевую цель своего создания. Тестировщики используют их, чтобы убедиться, что их тестирование тоже работает на эту цель.

Возьмем, к примеру, Google Sites — бесплатное приложение по созданию сайтов для открытых или закрытых сообществ. Как и у многих других пользовательских приложений, список практически всех нужных нам атрибутов уже написан внутри самого продукта. И верно: большинство приложений, имеющих что-то вроде страницы «С чего начать работу» или какие-нибудь рекламные тексты, уже определили атрибуты за вас. А если нет, то простой разговор со специалистом по продажам, просмотр коммерческих видео или презентаций даст вам необходимую информацию.

Важно понимать, что атрибуты лежат на поверхности. Если вам тяжело быстро перечислить их, то вы недостаточно знаете свой продукт, чтобы хорошо его тестировать. Знайте и любите свой продукт, и тогда составление списка атрибутов

станет минутным делом. Ну ладно, ладно, любить не обязательно, но и не запрещено законом.

The screenshot shows the 'Welcome to Google Sites' page. At the top, there's a navigation bar with 'Home' and 'Overview'. Below it, the 'Google Sites overview' section features several sections: 'Company Intranet', 'Team project', 'Employee profile', 'Classroom', and 'Student club'. Each section has a small thumbnail image and a brief description. Below these are 'Quick facts' with links to 'Single-click page creation', 'No HTML required', 'Make it your own', 'Get started with templates', 'Upload files and attachments', 'Embed rich content', 'Work together and share', 'Search with Google', and a footer note about file size limits. At the bottom, there are links to '©2011 Google - Terms of Service - Privacy Policy - Help Center'.

Рис. 3.2. Добро пожаловать на Google Sites

The screenshot shows the 'Google test analytics' interface. On the left, there's a sidebar with 'Sites-Test' selected, and a list of project settings: 'Project Spec', 'About Project', 'Attributes', 'Components', 'Capabilities', 'Risk', 'Overview', 'Imported Data', 'Tests', 'Bugs', 'Checkins', 'Data Settings', 'Data Sources', and 'Data Filters'. The main area is titled 'Sites-Test' and contains a 'Attributes' section. It lists several attributes with descriptions and checkboxes: 'searchable' (checkbox: 'The tests performed for this attribute are sufficient to verify its operation.'), 'sharing' (checkbox: 'The tests performed for this attribute are sufficient to verify its operation.'), 'quick' (checkbox: 'The tests performed for this attribute are sufficient to verify its operation.'), 'no technical knowledge' (checkbox: 'The tests performed for this attribute are sufficient to verify its operation.'), 'customization' (checkbox: 'The tests performed for this attribute are sufficient to verify its operation.'), and 'rich content' (checkbox: 'The tests performed for this attribute are sufficient to verify its operation.'). At the bottom, there's a copyright notice '© 2011 Google'.

Рис. 3.3. Атрибуты Google Sites, записанные в GTA

НА ЗАМЕТКУ

Если вам тяжело быстро перечислить атрибуты, то вы недостаточно знаете свой продукт, чтобы хорошо его тестировать.

В Google мы можем фиксировать список атрибутов по-разному: можем использовать текстовые документы, электронные таблицы или специальную программу Google Test Analytics (GTA), созданную нашими предприимчивыми инженерами. Совершенно не важно, что именно вы будете использовать, главное, чтобы атрибуты были записаны.

C — значит Component

Вы уже составили список атрибутов? Добро пожаловать в мир компонентов! Компоненты — это «существительные» нашего продукта. Это наши кирпичи, из которых построена вся система. Компоненты — это, к примеру, корзина и система оформления заказов в интернет-магазине, это возможность редактирования и печати в текстовом процессоре. Иными словами, компоненты — это ключевые части кода, которые делают программу тем, чем она является. Собственно, это и есть те вещи, которые тестировщикам поручено тестировать!

НА ЗАМЕТКУ

Компоненты — наши кирпичи, из которых построена вся система, это ключевые части кода, которые делают программу тем, чем она является.

Обычно за компонентами далеко ходить не надо, часто они уже определены в документе архитектуры системы. Компоненты в крупных системах — это большие прямоугольники на архитектурных диаграммах, а их названия часто появляются в тегах в багтрекинговой системе или явно указываются в проектной документации. В системах поменьше компонентами являются классы и объекты в коде. В любом случае можно подойти к разработчику и спросить, с какими компонентами он работает. Вам останется только зафиксировать список, и дело вместе с компонентами в шляпе.

Как и в случае с атрибутами, уровень детализации имеет значение. Слишком много подробностей — информация станет перегруженной, а отдача снизится. Слишком мало подробностей — станет непонятно, зачем вы вообще делали эту работу. Список должен быть небольшим. Получилось десять пунктов? Отлично! Двадцать? Многовато, если только ваша система не очень большая. Второстепенные подробности можно и пропустить, так как они или являются частью другого компонента, или не так уж и важны для конечного пользователя.

В самом деле, составить список атрибутов или компонентов — это дело нескольких минут. Если вам тяжело это сделать, то у вас однозначно слишком мало

знаний о продукте и нужно потратить время на его изучение. Продвинутый пользователь проекта даже спросонья немедленно перечислит список атрибутов продукта, так же как любой участник проекта, обладающий доступом к исходному коду и документации, легко перечислит список компонентов. Для нас тестирующий — это опытный пользователь и подкованный участник проекта одновременно.

Главное, не пытайтесь охватить все за один раз. Смысл АСС-анализа в том, чтобы подготовиться к работе быстро, а затем в ходе итераций отточить результат. Если вы пропустили важный атрибут, то вы наткнетесь на него, составляя список компонентов. Когда вы дойдете до описания возможностей продукта, забытые компоненты и атрибуты обязательно всплынут.

The screenshot shows the 'Components' section of the Google Test Analytics (GTA) interface. The left sidebar includes links for 'Project Spec', 'About Project', 'Attributes', 'Components' (selected), 'Capabilities', 'Risk Overview', 'Imported Data', 'Tests', 'Bugs', 'Checkins', 'Data Settings', 'Data Sources', and 'Data Filters'. The main content area is titled 'Components' and contains sections for 'Nav bar', 'Sitemap', 'Settings', 'Page view', 'Audit trail', and 'Search'. Each section has a text input field for 'Enter description of this component...' and a checkbox labeled 'The tests performed for this attribute are sufficient to verify its operation.' The URL at the bottom is https://test-analytics.googleplex.com/Testify_Mn88/1901/components.

Рис. 3.4. Компоненты Google Sites, записанные в GTA

C — значит Capability

Следующий этап АСС-анализа состоит в описании возможностей продукта, «глаголов» всей системы, действий, которые она выполняет по запросу пользователя. Это реакция программы на ввод данных, ответы на запросы и все операции, которые совершает приложение. По сути, пользователь выбрал этот продукт именно из-за возможностей: ему нужно что-то сделать, и ваше приложение может это сделать.

НА ЗАМЕТКУ

Возможности — это действия системы, которые она выполняет под руководством пользователя. Это реакция программы на ввод данных, ответы на запросы и все операции, которые выполняет приложение по запросу пользователя.

Возьмем, к примеру, Chrome. Он отображает веб-страницы, воспроизводит Flash-файлы, синхронизируется между клиентами и скачивает документы. Все это и многое другое составляет список возможностей браузера Chrome. Или возьмем интернет-магазин: он может найти товар и оформить заказ — это и есть его возможности. Проще говоря, если приложение может выполнить какую-то задачу, это мы и назовем его возможностью.

Возможности лежат на пересечении атрибутов и компонентов. То есть компоненты выполняют какую-то функцию для того, чтобы соответствовать атрибуту продукта, а результатом станет предоставление пользователю *возможности*. Давайте вернемся к Chrome для наглядности: Chrome *быстро* отображает страницы, он *безопасно* воспроизводит Flash-файлы. Если ваш продукт делает что-то, что не является пересечением компонента и атрибута, то, скорее всего, это что-то не имеет значения и даже есть смысл спросить, зачем это вообще нужно. Возможность, которая не работает на ключевую цель продукта, может дать потенциальные точки сбоев, поэтому лучше всего сбросить этот балласт.

Правда, есть вариант, что эта загадочная возможность все же имеет право на существование, просто вы этого не знаете, а значит — плохо изучили продукт. Тестировщику недопустимо не понимать суть происходящего — это вопрос профессиональной квалификации. Если хотя бы один из инженеров проекта лучше всех понимает, какие именно возможности продукт предоставляет пользователю, то этот инженер однозначно тестировщик.

Вот пример нескольких возможностей интернет-магазина:

- **Добавить/удалить товар из корзины.** Это возможность компонента «Корзина», когда он пересекается с атрибутом интерфейса «интуитивный».
- **Получить данные кредитных карт и верифицировать.** Это возможность компонента «Корзина», когда он пересекается с атрибутами «удобный» и «интегрированный» (речь об интеграции с платежной системой).
- **Обработать финансовые операции с помощью HTTPS.** Это возможность компонента «Корзина», когда он пересекается с атрибутом «безопасный».
- **Рекомендовать клиенту товары, основываясь на просмотренных им продуктах.** Это возможность компонента «Поиск», когда он пересекается с атрибутом «удобный».
- **Высчитать стоимость доставки.** Это возможность компонента «Интеграция с почтовой службой UPS», когда он пересекается с атрибутами «быстрый» и «безопасный».

- **Выводить информацию о наличии товара.** Это возможность компонента «Поиск», когда он пересекается с атрибутами «удобный» и «точный».
- **Отложить товар.** Это возможность компонента «Корзина», когда он пересекается с атрибутом «удобный».
- **Искать товар по ключевым словам, артикулу или категории.** Это возможность компонента «Поиск», когда он пересекается с атрибутами «удобный» и «точный». В общем случае мы рассматриваем каждую категорию поиска как отдельную возможность.

Разумеется, количество возможностей продукта может быть большим. Если вам кажется, что вы перечислили все, что вы *можете* протестировать, — ура, вы освоили АСС-анализ. Ведь основная задача этого процесса — кратко и быстро перечислить главные возможности системы, работоспособность которых надо проверить.

Возможности обычно отображают представление пользователя о том, что система может делать. То есть если краткость правит бал в составлении списка атрибутов и компонентов, то список возможностей может быть очень большим, ведь они описывают все, что может делать система. Чем сложнее функциональность приложения, тем длиннее будет список ее возможностей.

У систем, над которыми мы работаем в Google, список возможностей доходит до ста и более пунктов (например, у Chrome OS их больше трех сотен), у более простых приложений могут быть десятки возможностей. Конечно, есть продукты с небольшим количеством возможностей, тогда их могут протестировать разработчики и первые пользователи. Поэтому если вы тестируете приложение, у которого меньше двадцати возможностей, задайтесь вопросом: а ваша работа здесь действительно нужна?

Самый важный аспект возможностей продукта — это то, что мы можем их протестировать. Не зря мы пишем возможности глаголами — ведь это активный признак субъекта. Глагол требует действий с нашей стороны — мы должны написать тест-кейсы, которые проверят, насколько точно реализована каждая возможность в системе и найдет ли пользователь свое взаимодействие с такой системой полезным. Позднее мы подробно обсудим, как возможности превращаются в элегантные тест-кейсы.

НА ЗАМЕТКУ

Самый важный аспект возможностей продукта — это то, что мы можем их протестировать.

Насколько общими должны быть возможности — предмет ожесточенных споров тестировщиков в Google. По определению, возможности не могут фиксироваться излишне детально, так как одна возможность может описывать любое количество сценариев пользователя. Например, в интернет-магазине, который

мы приводили в пример, возможности не уточняют, какой предмет положили в корзину, и не описывают результат, который мы получим при конкретном поиске. Они только говорят нам о том, какие действия может выполнить пользователь. Такое обобщение делается намеренно, так как бессмысленно тратить время на тонны документации, если мы не собираемся это тестировать. Мы не можем протестировать все возможные варианты поиска и состояния корзины, поэтому напишем тест-кейсы только на основе тех вариантов, которые действительно будем тестировать.

Возможности не должны быть аналогами тест-кейсов — точных значений и данных у них нет. Например, для возможности «пользователь может что-то купить» тест-кейс будет включать «что именно он покупает». Возможности предусматривают наличие тестов, направляют их в нужное русло, но сами таковыми не являются.

Вернемся к примеру с Google Sites: обратите внимание на рис. 3.5, где в столбцах представлены атрибуты, а в строках — компоненты. Так мы связываем атрибуты с компонентами. Как вы видите, далеко не каждый компонент работает со всеми атрибутами, поэтому появляются пустые ячейки. В Chrome только некоторые компоненты связываются с атрибутами «быстрый» и «безопасный». Пустая ячейка на пересечении «атрибут–компонент» означает, что тестировать эту пару не нужно.

The screenshot shows the Google Test Analytics (GTA) interface. On the left, there's a sidebar with navigation links like 'Project Spec', 'About Project', 'Attributes', 'Components', 'Capabilities', 'Risk Overview', 'Imported Data', 'Tests', 'Bugs', 'Checkins', 'Data Settings', 'Data Sources', and 'Data Filters'. The main area has a title 'Sites-Test' and a sub-section 'Capabilities'. A sub-sub-section titled 'Capabilities by Attribute and Component' is expanded, showing a table with columns: searchable, sharing, quick, no technical knowledge, customization, and rich content. Rows represent components: Nav bar, Sitemap, Settings, Page view, Audit trail, and Search. The 'sharing' column for 'Page view' contains a value '3' with a cursor icon over it. Below the table, a section titled 'Page view is sharing' lists three options: 'Accessible online to anyone (if you want)', 'Share page management (crud) with someone else', and 'Page view not loading at all'. At the bottom right of the interface, it says '© 2011 Google'.

	searchable	sharing	quick	no technical knowledge	customization	rich content
Nav bar	1	1		1		
Sitemap		1	1	1	1	
Settings	1	11	1			
Page view	1	3	1	1	4	12
Audit trail	1	2	1	1		
Search		2	1		1	1

Рис. 3.5. Связь компонентов и атрибутов в GTA

Каждая строка или столбец в таблице — это срез функциональности системы, объединенный общим признаком. Можно разделить таблицу по строкам или по

столбцам — и перед вами готовые планы тестовых сессий. Тест-менеджер может раздать командам разные строки или провести целенаправленную атаку на баги в конкретном столбце. Такое деление отлично подходит для исследовательского тестирования: вручив тестировщикам разные столбцы и колонки, вы избавитесь от совпадений и обеспечите более высокое покрытие.

Числовые значения в ячейках показывают количество возможностей, которые может предложить компонент для выполнения атрибута. Чем число выше, тем больше тестов связано с таким пересечением. Например, компонент «Просмотр страницы» взаимодействует с атрибутом «доступный» в трех возможностях:

- сделать документ доступным для сотрудников;
- дать возможность сотрудникам редактировать документ;
- показывать положение сотрудника на странице.

Итак, эти возможности проясняют моменты, которые нужно протестировать для пары «Просмотр страницы» и «доступный». Мы можем написать тест-кейс для каждой из них или протестировать комбинацию возможностей, объединив их в более крупный сценарий использования или тестовый сценарий.

Умение описать хорошие возможности требует определенного навыка. Вот несколько самых важных свойств возможностей, знание которых поможет при работе:

- Возможность должна быть представлена как *действие* и описывать выполнение пользователем одной задачи в тестируемом приложении.
- Возможность должна предоставлять тестировщику достаточно информации, чтобы он понял, какие переменные надо учитывать при написании тест-кейса. Например, дана возможность — «Обработать финансовые операции с помощью HTTPS». В данном случае тестировщик должен понимать, какие финансовые операции может выполнить система, какой механизм будет проверять осуществление операции через HTTPS. Да, работа немалая! Если вы думаете, что какие-то финансовые операции могут быть упущены, скажем, новым тестировщиком, то продублируйте эту возможность для разных типов операций. Опять же, если команда тестирования собаку съела на HTTPS, то общей формулировки будет вполне достаточно. Не усложняйте себе задачу и позволяйте возможностям оставаться общими, оставьте тест-кейсам и исследовательским тестировщикам самим определить нужный уровень детализации¹.

¹ Позволить тестировщику детализировать возможность — здравая мысль: становится больше вариантов интерпретации функциональности и их преобразований в тестовые примеры, а это, в свою очередь, улучшает покрытие.

- Возможность должна стыковаться с другими возможностями. Сценарии использования или пользовательские сценарии должно быть можно представить как серию возможностей. Если это сделать нельзя, то в вашей системе какие-то возможности отсутствуют или представлены слишком общими понятиями.

Преобразование набора возможностей в пользовательские истории — это необязательный шаг, но он способен сделать всю систему тестирования более гибкой. В Google есть несколько команд, которые предпочитают более общие пользовательские истории частным тест-кейсам, когда работают с внешними подрядчиками или при организации исследовательского краудсорс-тестирования. Почему? Слишком конкретные тест-кейсы, выполняемые сторонним тестировщиком многократно, вызывают скуку, становятся рутиной, в то время как пользовательские истории дают больше свободы действий, делают процесс тестирования творческим и интересным, защищают от ошибок, которые можно сделать, занимаясь механическим процессом, уже набившим оскомину.

Что бы вы ни выбрали своей целью — создание тест-кейсов, пользовательских историй, а может, и того и другого, — используйте общие правила для трансформации возможностей в тест-кейсы. Имейте в виду, что это просто направляющие, а не абсолютные утверждения.

- Каждая возможность должна быть связана хотя бы с одним тест-кейсом. Если она была записана, значит достаточно важна для того, чтобы ее протестировать.
- Многие возможности требуют более одного тест-кейса. Каждый раз, когда во входной информации есть отклонения, последовательности ввода, системные переменные, тест-кейсов нужно делать несколько. Атаки, описанные в книге «How to Break Software», и туры в «Exploratory Software Testing» здорово описывают принципы выбора тестовых примеров и подход к выбору данных, которые легко превращают возможность в тест-кейс, вылавливающий баг.
- Не все возможности равны. Есть те, которые важнее других. На следующем шаге процесса возможности связываются с рисками и определяют степень их важности.

Завершив АСС-анализ, мы знаем все, что мы *могли бы* протестировать при неограниченном бюджете и времени. Но поскольку часто не хватает то одного, то другого, будет полезно выделить главное. В Google такая расстановка приоритетов называется *анализом рисков*, и это будет нашей следующей темой.

Пример: определение атрибутов, компонентов и возможностей Google+

ACC-анализ можно быстро выполнить в текстовом документе, таблице или даже на салфетке! Ниже следует сокращенный вариант ACC-процесса для Google+.

Атрибуты Google+ (список сформирован на основе наблюдения за дискуссией руководства Google).

- Социальный: позволяет пользователю обмениваться информацией и мыслями.
- Выразительный: пользователи используют возможности продукта для самовыражения.
- Простой: пользователь легко понимает, как сделать то, что он хочет.
- Релевантный: показывает только ту информацию, которая интересует пользователя.
- Расширяемый: интегрируется с другими ресурсами Google, сторонними сайтами и приложениями.
- Конфиденциальный: данные пользователя не будут открытыми.

Компоненты Google+ (получены из архитектурной документации).

- Профиль: информация и настройки текущего пользователя.
- Люди: профили людей, с которыми связан пользователь.
- Лента: ранжированная лента сообщений, комментариев, оповещений, фотографий и т. д.
- Круги: группы контактов («друзья», «коллеги» и т. д.).
- Оповещения: обозначения упоминания пользователя в сообщении.
- Интересы, или «+1»: обозначения материалов, которые понравились пользователю.
- Записи: сообщения о записях пользователей и их кругов.
- Комментарии: комментарии к сообщениям, фотографиям, видеороликам и т. д.
- Фотографии: фотографии, загруженные пользователями и их друзьями.

Возможности Google+.

Профиль:

- Социальный: обмениваться профилями и предпочтениями с друзьями и контактами.
- Выразительный: создавать онлайн-версию самих себя.
- Выразительный: взаимодействовать с Google+ по-своему.
- Простой: легко вводить, обновлять и распространять информацию.
- Расширяемый: передавать данные профилей приложениям с соответствующими правами доступа.
- Конфиденциальный: сохранять свои данные конфиденциальными.
- Конфиденциальный: передавать данные только одобренным пользователям и приложениям.

Люди:

- Социальный: связываться с друзьями, коллегами и членами своих семей.
- Выразительный: легко различать профили других пользователей.
- Простой: удобно управлять контактами пользователя.
- Релевантный: фильтровать свои контакты по своим критериям.
- Расширяемый: передавать контактную информацию службам и приложениям, имеющим необходимые разрешения.
- Конфиденциальный: предоставлять данные о контактах пользователя только сторонам с соответствующими разрешениями.

Лента:

- Социальный: информировать пользователей об обновлениях их социальных сетей.
- Релевантный: фильтровать те обновления, которые интересуют пользователя.
- Расширяемый: передавать обновления ленты службам и приложениям.

Круги:

- Социальный: группировать контакты на основании социального контекста.

- Выразительный: создавать новые круги на основе контекста пользователя.
- Простой: удобно добавлять, обновлять и удалять контакты из кругов.
- Простой: удобно создавать и изменять круги.
- Расширяемый: передавать данные о кругах службам и приложениям.

Оповещения:

- Простой: показывать оповещения кратко.
- Расширяемый: отправлять оповещения другим службам и приложениям.

Видеочат:

- Социальный: приглашать свои круги в видеочат.
- Социальный: открыть свой видеочат публике.
- Социальный: оповещать других пользователей в своих лентах о видеочатах.
- Простой: создавать видеочат и принимать в нем участие в несколько кликов.
- Простой: отключить в один клик видео- и аудиоданные.
- Простой: приглашать дополнительных пользователей в существующий видеочат.
- Выразительный: посмотреть, как видеочат будет выглядеть для других.
- Расширяемый: общаться в текстовом чате во время видеочата.
- Расширяемый: включать видеоролики с YouTube в видеочат.
- Расширяемый: настраивать устройства в Настройках.
- Расширяемый: участвовать в видеочатах без веб-камеры, используя аудиоканал.
- Конфиденциальный: ограничивать доступ в видеочат только для приглашенных гостей.
- Конфиденциальный: оповещать только приглашенных гостей о видеочате.

Записи:

- Выразительный: выражать свои мысли.
- Конфиденциальный: ограничивать сообщения выбранной аудиторией.

Комментарии:

- Выразительный: выражать свое мнение с помощью комментариев.
- Расширяемый: передавать данные комментариев для использования другими службами и приложениями.
- Конфиденциальный: ограничивать сообщения выбранной аудиторией.

Фотографии:

- Социальный: делиться фотографиями с контактами и друзьями.
- Простой: легко загружать новые фотографии.
- Простой: легко импортировать фотографии из других источников.
- Расширяемый: интегрироваться с другими фотослужбами.
- Конфиденциальный: ограничивать доступ к фотографиям только для выбранной аудитории.

На рис. 3.6 приведены результаты ACC-анализа в форме электронной таблицы.

The screenshot shows a Google Sheets spreadsheet titled "ACC Grid". The columns represent different attributes or components of Google+, and the rows represent specific features. The columns are labeled A through G, and the rows are numbered 1 through 18. The data is organized into several sections:

- Profile:** Row 2. Contains 1 point under Social, 1 point under Expressive, and 1 point under Relevant.
- People:** Row 3. Contains 1 point under Social, 1 point under Expressive, and 1 point under Relevant.
- Perfect Stream:** Row 4. Contains 1 point under Social.
- Circles:** Row 5. Contains 1 point under Social, 1 point under Expressive, and 1 point under Relevant.
- Notifications:** Row 6. Contains 1 point under Social.
- Hangouts:** Row 7. Contains 1 point under Social, 1 point under Expressive, and 1 point under Relevant.
- Posts:** Row 8. Contains 1 point under Social.
- Comments:** Row 9. Contains 1 point under Social.
- Photos:** Row 10. Contains 1 point under Social, 1 point under Expressive, and 1 point under Relevant.

Each row also includes a column for "Extensibility" and "Privacy" with corresponding notes. The "Privacy" notes include:

- Profile: "Enables a user's to keep their private data, private, and only share it with the appropriate parties."
- People: "Shares contact data with authorized parties and applications."
- Perfect Stream: "Keeps data about a user's contacts private."
- Circles: "Shares stream updates to services and applications."
- Notifications: "Shares data about circles for use to services and applications."
- Hangouts: "I don't think these are primarily internal to the user, so privacy concerns are minimal."
- Posts: "Only invited guests can see posts from a hangout."
- Comments: "Comments are restricted to the intended audience."
- Photos: "Photos are restricted so that they're only visible to the intended audience."

Рис. 3.6. Электронная таблица ACC для Google+

А на рис. 3.7 эти же данные представлены в другой форме.

A	B	C	D	E	F	G
1 Attributes/Components	Social	Expressive	Easy	Relevant	Extensible	Private
Profile	1 - Share profiles and preferences with friends and contacts.	1 - Users can create an online version of themselves. 2 - Personalize their experience with Google+	1 - Easy to enter and update information, and have it propagate.	0	1 - Serves profile information to applications with appropriate access.	1 - Enables a user's to keep their private data, private, and only share it with the appropriate parties. 2 - Share data only with approved/appropriate parties. 3 - Keeps data about a user's contacts private.
People	1 - Users can connect with users' friends, coworkers, and family.	1 - Profiles of other users are personalized and easily distinguishable	1 - Provides tools to easily manage a user's contacts.	1 - Users can filter their contacts based on criteria for relevance.	1 - Serves contact data to authorized services and applications.	0
Stream	1 - Informs the user of updates from their social network.	0	0	1 - Filters for updates the user would be interested in.	1 - Serves stream updates to services and applications	3
Circles	1 - Groups contacts into circles based on social context.	1 - Creates new circles that are customized to the user's preferences.	1 - Facilitates adding, updating, removing, contacts to/from Circles 2 - Facilitates creating and modifying circles.	0	1 - Serves data about circles for use to services and applications	3
Notifications	0	0	1 - Presents notifications concisely.	0	1 - Posts notifications for use by other services and applications.	3 - I don't think these are completely internal to the user, so privacy concerns are minimal.
Hangouts	1 - Users can invite their circles to hangout. 2 - Users can open hangouts to the public. 3 - Others are notified of hangouts they access to in their stream.	1 - Before joining a hangout users can preview how they will appear to others.	1 - Hangouts can be created and participated in within a few clicks. 2 - Video and audio inputs can be disabled in a single click. 3 - Additional users can be added to an existing hangout.	0	1 - Users can chat through text while in a hangout. 2 - Videos from YouTube can be added to a hangout. 3 - Details can be configured and adjusted in Settings 4 - Users without a webcam can participate in hangouts through audio. 1 - Participants on stream posts for use by other services and applications.	1 - Only invited guests can access a hangout. 2 - Only invited guests are notified of a hangout.
Posts	0	0	0	0	1 - Posts are restricted to the intended audience.	1 - Posts are restricted to the intended audience.
Comments	0	1 - Expresses the thoughts of the user through the stream 1 - Expresses the thoughts of the user through comments.	0	0	1 - Comments are restricted to the intended audience	2 - Only invited guests are notified of a hangout.
Photos	1 - Users can share their photos with their contacts and friends.	0	1 - Users can easily upload relevant photos 2 - Users can easily import photos from other sources.	0	1 - Posts data on Comments for use by other services and applications. 1 - Integration with other Photo services	1 - Photos are restricted so that they're only visible to the intended audience.
10						
11						
12						
13						
14						
15						
16						

Рис. 3.7. Таблица ACC для Google+

Риск

Риски повсюду. Дома, на дорогах, на работе. Все, что мы делаем, включает в себя элемент риска, и разработка ПО — не исключение. Чтобы обезопасить свою жизнь, мы покупаем безопасные автомобили и водим осторожно. На совещаниях мы следим за своими словами и стараемся попасть на проекты, которые сможем выполнить, — все это для того, чтобы уменьшить риск остаться без работы.

А что сделать, чтобы уменьшить риск выпуска некачественного программного продукта? Как предотвратить опасность возникновения сбоев в выпускаемых нами программах, как уберечь нашу компанию от вероятного ущерба ее репутации? Разумеется, отличный способ минимизировать риски — не выпускать программу. Нет программы — нет риска. Но секрет вот в чем: мы получаем прибыль от выпуска программного обеспечения, если хорошо просчитываем риски.

Именно «просчитываем», не «вычисляем» — мы не гонимся за математической точностью. Мы ходим по тротуарам, а не гуляем по проезжей части, потому что так безопаснее, а вовсе не потому, что чьи-то расчеты показали, будто так мы снизим риск попасть под машину на 59%. Мы покупаем машины с подушками безопасности не потому, что помним точные статистические данные повышения выживаемости, а просто это всем известно: подушка безопасности снижает риск

разбить голову о руль. Все. Работа по снижению рисков может быть максимально эффективной и безо всяких трудоемких математических расчетов. Такая работа называется *анализом рисков*.

Анализ рисков

В осознании и просчитывании рисков, которые следует учитывать при тестировании, главным помощником был и остается здравый смысл. Переключим здравый смысл в режим «включено» и ответим на следующие важные вопросы:

- О каких событиях нам следует беспокоиться?
- Насколько такие события вероятны?
- Каким может быть от них ущерб для нашей компании?
- Каким может быть от них ущерб для будущих пользователей?
- Есть ли в программе защита от возникновения нежелательных событий?
- Какова вероятность того, что такая защита не сработает?
- Во сколько нам обойдется такой сбой защиты?
- Насколько сложно будет восстановиться после такого сбоя?
- Нежелательное событие может случиться вновь или эта проблема может быть одноразовой?

Нет резона использовать математический расчет при анализе рисков просто потому, что количество вероятных переменных настолько велико, что на их точный расчет уйдет больше сил, чем на саму минимизацию рисков. В Google мы сводим все риски к двум факторам: *частота сбоев* и *степень воздействия*. Для каждой возможности в разрабатываемом продукте тестировщики присваивают этим двум факторам простые значения. Именно простые, а не идеально точные. Нам ведь нужно просто определить, какие возможности следует тестировать и в каком порядке. Для этого достаточно понять, какая возможность несет в себе больший риск, а какая меньший. В этом нам здорово помогает GTA (рис. 3.8).

GTA использует четыре оценки частоты появления сбоев:

- **Очень редко:** трудно представить ситуацию, при которой возникнет проблема, но если такое случится, то ее решение будет простым.
- **Пример:** страница загрузки браузера Google Chrome¹. В основном содержимое этой страницы статично, автоматика там срабатывает только при

1 <http://www.google.com/chrome>

определении клиентской ОС. Если произойдет сбой в базовом коде HTML или в сценарии на странице, то его быстро обнаружит мониторинг кода.

- **Редко:** есть случаи, когда может произойти сбой. Однако из-за невысокой сложности процесса или редкого использования такое случается нечасто.

Пример: кнопка «Вперед» в Chrome. Эта кнопка используется, но гораздо реже, чем ее напарница кнопка «Назад». Так сложилось, что ее работа редко приводит к сбоям, а если такое все-таки случится, то проблема будет быстро перехвачена еще на этапе тестирования раннего выпуска нашими первыми пользователями.

- **Иногда:** такой сбой легко представить, хотя он и достаточно сложен технически, а возможность продукта, как мы считаем, будет популярна у пользователей.

Пример: Возможности Chrome Sync. Chrome синхронизирует закладки, темы, заполнение форм, историю просмотра страниц и другие данные пользователя между компьютерами. Есть много типов данных и платформ ОС, поэтому слияние изменений — задача непростая. Если произойдет сбой синхронизации данных, пользователь, вероятнее всего, это заметит. Синхронизация выполняется только в том случае, когда происходят изменения в синхронизируемых данных — например, пользователь добавил новую закладку.

The screenshot shows the Google Test Analytics (GTA) interface for the Google+ project. The left sidebar lists various project sections like Project Spec, Risk Overview, and Data Settings. The main area displays a table of failure statistics for Hangouts, Posts, Comments, and Photos. A modal window titled 'Hangouts is Easy' is open, detailing a specific risk: 'Additional users can be added to an existing hangout.' It includes fields for Name (set to 'new label'), Description (empty), Frequency of Failure (Occasionally), Impact (Considerable), Attribute (Easy), and Component (Hangouts). Below this, two other risks are listed: 'Video and audio inputs can be disabled in a single click.' and 'Hangouts can be created and participated in within a few clicks.'

Рис. 3.8. Оценка риска по частоте и воздействию в GTA для Google+

- **Часто:** если в возможности, которая входит в часто используемую фичу, регулярно происходят сбои.

Пример: отображение веб-страниц. Основная задача браузера — отобразить HTML, CSS и код JavaScript любого происхождения и качества. Даже если исходный загружаемый код будет технически неправильным, пользователь решит, что проблема в браузере. Риск сильно возрастает, если мы рассматриваем сайт с высоким уровнем трафика. Проблемы отображения не всегда могут быть обнаружены пользователем: визуально элемент может съехать, но тем не менее остается рабочим. Или он может исчезнуть совсем, но тогда как пользователь узнает, что он вообще был?

Итак, тестировщик выбирает одно из этих значений для каждой возможности. Мы специально сделали четыре варианта оценки, чтобы тестировщик не вздумал постоянно выбирать средний вариант. Здесь нужно серьезно подумать.

Для оценки воздействия мы используем такой же упрощенный метод, с таким же количеством вариантов (снова возьмем браузер Chrome для примера).

Воздействие может быть:

- **Минимальным:** сбой, который пользователь может не заметить.

Пример: есть такой дополнительный сервис Chrome Labs. Его функциональность необязательна для работы, и сбой при загрузке страницы chrome://labs затронет лишь нескольких пользователей. Эта страница содержит дополнительные, экспериментальные фичи Chrome, о которых большинство людей даже не знают. Сами фичи снабжены пометкой «Используйте на свой страх и риск». Проще говоря, компания не несет ответственности за сбои, но зато и угроз для работы браузера нет.

- **Небольшим:** сбой, который может вызвать раздражение у пользователя. Если случается — механизмы повтора и восстановления легкодоступны.

Пример: нажмем на кнопку «Обновить». Если она не обновляет страницу, то можно заново ввести URL-адрес или открыть новую вкладку и попробовать ввести его там, а можно просто перезапустить браузер. Самые худшее в этой ситуации — раздражение пользователя.

- **Существенным:** сбой блокирует выполнение пользовательских сценариев.

Пример: расширения Chrome. Если пользователь установил расширения в свой браузер, а в новой версии Chrome возник сбой при загрузке этих расширений, это провал.

- **Максимальным:** сбой нанесет удар по репутации продукта и заставит пользователей отказаться от работы с ним.

Пример: механизм автообновления Chrome. Если эта возможность отвалится, то браузер лишится важных обновлений безопасности или вовсе прекратит работать.

Иногда сбои приводят к разным последствиям у пользователя и компании. Допустим, перестал работать рекламный баннер. Проблема ли это для пользователя? Нет, он даже может не заметить. Проблема ли для Google? Да, конечно. Поэтому, анализируя риски, указывайте, кто пострадает.

Данные, которые присвоил рискам тестировщик, можно наложить на готовую таблицу «атрибут/компонент» для Google Sites. Таким образом мы получим тепловую карту рисков (рис. 3.9).

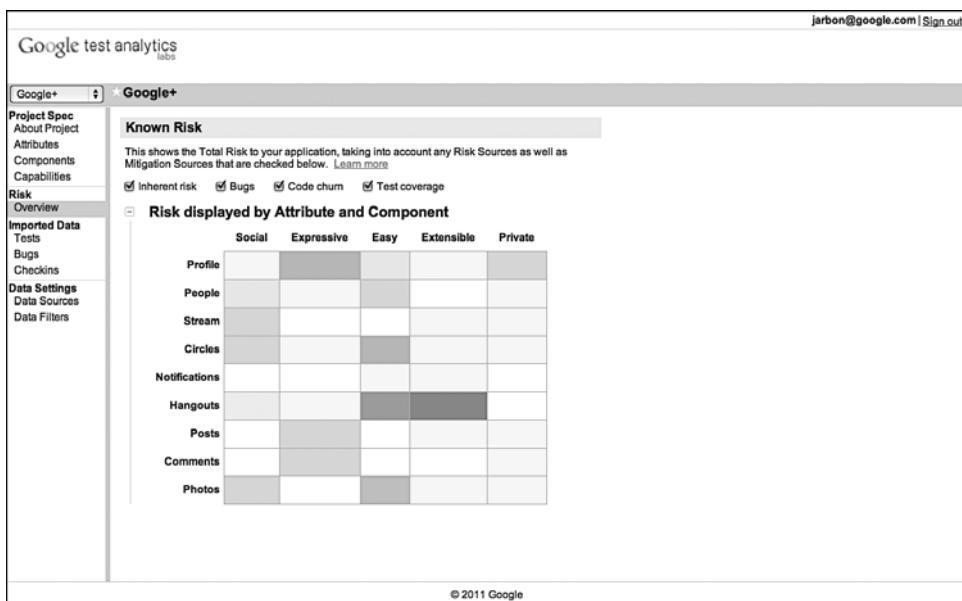


Рис. 3.9. Тепловая карта рисков для таблицы «атрибут/компонент» (ранняя версия Google+)

Ячейки окрашиваются в красные, желтые или зеленые цвета в зависимости от уровня риска компонентов. Очень просто вычислить примерный уровень риска для каждого значения — мы просто усредняем риски его возможностей. Эта карта была сделана в GTA, но с той же задачей справится и обыкновенная электронная таблица.

Такая диаграмма показывает возможности продукта, которые можно тестировать, вместе с присвоенными значениями рисков их сбоев. Трудно проставить эти значения объективно, тем более что у тестировщиков весьма специфическая точка зрения на продукт. Чтобы подстраховаться, мы опрашиваем и других заинтересо-

ванных в проекте лиц. Вот список тех людей, которые могут помочь с определением рисков, и несколько советов, как работать с их мнениями.

- **Разработчики.** Большинство разработчиков присвоят максимальное значение риска функции, которую писали они. Естественно, они хотят, чтобы их код был протестирован! Каждый кулик хвалит свое болото, но наш опыт подсказывает, что «кулик» часто переоценивает те функции, за которые он отвечает.
- **Руководитель проекта.** Удивительно, но руководители проектов тоже люди, и их оценка тоже может быть субъективной. Для них наиболее важными считаются те функции, которые позволяют продукту выделиться на рынке и стать хитом.
- **Специалисты по продажам.** Эти парни зарабатывают на привлечении клиентов, поэтому их оценка возможностей смешена к тем, которые помогают продукту хорошо смотреться в демоверсии.
- **Директора и вице-президенты.** Топ-менеджеров обычно интересуют те возможности продукта, которые качественно отличают его от главных конкурентов.

Итак, у всех мнений есть погрешность. Чтобы бороться с этим, мы опрашиваем всех заинтересованных в проекте людей по отдельности и просим оценить риски по описанным выше двум факторам. Их не так легко уговорить на эту работу, но мы, кажется, нашли успешную стратегию. Вместо того чтобы вдаваться в объяснения процесса и упрашивать участников нам помочь, мы просто делаем все сами и показываем им уже готовую тепловую карту. Как только они видят наше мнение, они мгновенно выплескивают свое. Разработчики активно принимают участие, если понимают, что мы используем карту для расстановки приоритетов в тестировании. Так же ведут себя и руководители проектов, и менеджеры по продажам. Все они заинтересованы в его качестве.

Есть в этом подходе определенная сила. Когда мы определяем риски самостоятельно, мы несомненно приходим к варианту, с которым другие участники проекта не будут согласны. Так и есть, подготавливая для них наш вариант анализа рисков, мы даем им почву для споров. В этом идея. Мы не спрашиваем их о каких-то абстрактных понятиях, мы даем им конкретные выводы, которые можно оспорить. Люди не всегда могут сказать, каким должен быть правильный ответ, но легко скажут, каким он быть *не должен*. Такой хитрый подход приносит нам много правдивых данных для вычисления рисков.

Когда риски будут согласованы, можно приступать к их последовательному снижению.

Снижение рисков

Редко удается полностью устраниТЬ риски. Мы водим машину, хоть это и опасно, но ведь нужно добираться до работы? Вообще возможность несчастного случая не означает, что он обязательно произойдет, да и, скорее всего, ничего страшного не случится. Почему? Потому что своими действиями мы снижаем возможный риск. Например, не садимся за руль в нетрезвом состоянии и не водим в условиях недостаточной видимости. Таким образом мы *снижаем риски*.

В разработке программного продукта самое простое — избегать рискованных областей: чем меньше кода, тем меньше риск. Но кроме использования «топора и секиры», мы можем сделать еще много чего, чтобы снизить риски:

- Мы можем проработать пользовательские истории вокруг наиболее рискованных возможностей, определить самые безопасные пути и показать их разработчикам, чтобы те ввели в приложение больше ограничений.
- Мы можем написать регрессионные тест-кейсы, чтобы убедиться, что мы отловим повторные сбои.
- Мы можем написать и запустить тесты, подтверждающие необходимость добавить механизм восстановления и отката.
- Мы можем добавить средства контроля и сторожевой код для оперативного обнаружения сбоев.
- Мы можем добавить инструменты, которые будут отслеживать изменения в поведении продукта в его разных версиях. Мы получим сигнал, если возникнет регрессионный баг.

Конкретное решение зависит от особенностей приложения, от ожиданий пользователя в отношении его безопасности и надежности. Как тестирущики, мы, конечно, можем быть вовлечены и в процесс снижения рисков, но мы безусловно вовлечены в процесс их выявления. Мы начинаем с приоритизации возможностей, отмеченных в таблице красным. Мы хотим тестировать в порядке уменьшения рисков. Это важно: *если не можешь протестировать все — протестируй сначала самое важное*. А самое важное — это то, что больше всего подвержено самым серьезным рискам.

В некоторых проектах именно тестирующих спрашивают о готовности продукта к выпуску. Хорошему тестирующему достаточно бросить взгляд на тепловую карту, чтобы определить, стоит еще подержать продукт в духовке или пора подавать его на стол. Если речь о запуске экспериментального Google Labs, то наличие красных зон риска не так существенно, если они не относятся к безопасности, конечно. А если это выпуск новой версии Gmail, тогда даже желтые зоны пред-

ставляют серьезную опасность. Такая простая цветовая градация понятна всем, даже топ-менеджерам.

Опасения по поводу рисков со временем спадают, а большой объем успешно проведенного тестирования — это хороший признак того, что риски на приемлемом уровне. Здесь мы выигрываем от того, что связываем тест-кейсы с отдельными возможностями продукта, а затем и с атрибутами и компонентами в таблице рисков. Для этого дела идеально подходит «АСС-анализ», и вот почему мы создали этот инструмент именно таким.

Тест-план за десять минут по рецепту Джеймса Уиттакера

Любая задача в разработке ПО, которую можно решить за десять минут, считается простой или не заслуживающей внимания. Предположим, что мы верим в это, — тогда что мы можем сказать о планирования тестирования? Конечно же, то, что оно занимает более десяти минут. Когда я работал директором по тестированию в Google, я руководил несколькими командами, которые создавали огромное количество тест-планов. Ответы на вопрос о том, сколько времени займет его составление, могли быть такими: «завтра», «к концу недели» и лишь пару раз — «к концу дня» (если задача озвучивалась рано утром). О'кей, примем к сведению, что составление тест-плана занимает некоторое количество часов, а то и дней.

Стоит ли такая работа усилий — это уже совсем другая история. Я вижу десятки тест-планов, которые пишут мои команды, и каждый раз это мертворожденные документы — они создаются, рецензируются, обновляются один или два раза (если повезет), а потом уверенно откладываются в долгий ящик, как только проект начинает идти не так, как это было предусмотрено. Возникает вопрос: если план не стоит того, чтобы его обновлять, стоило ли его создавать?

Иногда тест-план нежизнеспособен потому, что содержит слишком много или, наоборот, слишком мало подробностей. Или он способствовал началу работы, а вот процессу — уже нет. И снова вопрос знатокам: стоило ли создавать документ с ограниченной или постоянно уменьшающейся ценностью?

Некоторые тест-планы содержат настолько очевидную информацию, что ее и документировать-то не стоило. Мы просто зря тратим время. Давайте посмотрим правде в глаза: у нас проблема с тест-планами.

Чтобы справиться с этим, я придумал для своей команды простое задание: написать тест-план за десять минут. Если уж он и имеет какую-то ценность, то давайте доберемся до нее как можно скорее.

Когда у вас есть всего десять минут для решения задачи, каждая секунда становится значимой. В этом моя основная идея: ограничение во времени заставляет отсекать при планировании всю шелуху и концентрироваться только на важных моментах. Делайте только то, что *абсолютно необходимо*, оставьте подробности исполнителям тестов. Я хотел покончить с порочной практикой написания нежизнесспособных тест-планов, и это упражнение показалось мне верным.

Однако я ничего этого я не говорил участникам эксперимента. Я просто сказал: «Вот приложение, составьте тест-план не более чем за десять минут». Имейте в виду, что эти люди получали зарплату за то, что они выполняли мои задачи. И все же я предполагал, что они испытывали ко мне определенноеуважение, а следовательно, знали, что я не поручу им невыполнимую задачу.

Они могли потратить некоторое время на знакомство с приложением, но, так как речь шла о приложениях, которые они используют каждую неделю (Google Docs, App Engine, Talk Video и т. д.), я дал им совсем немного времени на это.

Во всех случаях команды изобретали методы, схожие с методами АСС-анализа. Они оформляли решения в форме таблиц и списков, не используя большие объемы текста. То есть предложениям — да, абзацам текста — нет. Они не тратили время на форматирование текста, не вдавались в излишние объяснения. У всех тест-планов было одно общее — команды документировали возможности. Они признали, что это было лучшим решением, куда потратить весьма ограниченное время.

О'кей, ни одна команда не завершила тест-план вовремя. Тем не менее они успели за десять минут пройтись по атрибутам и компонентам и начали вычленять возможности исследуемого продукта. К концу дополнительных двадцати минут большинство моих подопытных записали довольно большой набор возможностей, который мог бы служить отличной отправной точкой при создании тест-кейсов и пользовательских историй.

Мне кажется, что эксперимент удался. Я выделил им десять минут, хотя ориентировался на час. В итоге за полчаса было выполнено 80% работы. Разве этого недостаточно? Мы точно знаем, что не будем тестировать все, ну и зачем нам все документировать? Мы отлично знаем, что в ходе тестирования многие вещи (графики, требования, архитектура) будут изменяться. Настаивать на скрупулезной точности планирования, когда завершенность вовсе не требуется, не имеет смысла.

Восемьдесят процентов работы выполнено за тридцать минут или даже меньше. Вот это я называю десятиминутным тест-планом!

Напоследок о рисках

Google Test Analytics берет за основу описанные выше критерии оценки рисков («очень редко», «редко», «иногда», «часто»). Мы специально не хотим превращать анализ рисков в сложную задачу, иначе она не будет выполнена. Нас не интересуют точные математические подробности, потому что цифры мало что значат. Достаточно знать, что «А» рискованнее «Б», не обращая внимания на точное значение рисков. Простое знание, какая возможность рискованнее другой, позволит тест-менеджеру более эффективно распределять работу тестировщиков. А такие люди, как Патрик Коупленд, смогут легко решать, сколько тестировщиков нужно назначить в каждую команду разработки. Понимание рисков приносит пользу на уровне всей компании.

Анализ рисков — это самостоятельная научная область, уважаемая во многих отраслях. Мы используем упрощенную версию методологии, но это не мешает нам интересоваться новыми исследованиями, чтобы улучшить свой подход к тестированию. Если вы хотите узнать больше об анализе рисков, то начните со статьи «Управление рисками» в Википедии.

GTA помогает обозначить риски, а тестирование помогает их снизить. Тестировщик служит посредником в этом процессе. Он может выполнить внутренние тесты по некоторым наиболее рискованным направлениям или поставить задачу разработчикам и разработчикам в тестировании, чтобы они добавили регрессионные тесты. В его арсенале есть и другие инструменты: исследовательское тестирование, привлечение внутренних и бета-пользователей и силы внешнего сообщества.

В ответственности тестировщика знать все подверженные рискам области. Он должен стараться снизить риски любыми способами, которые ему подвластны. Вот несколько рекомендаций, которые мы считаем полезными в борьбе с рисками.

1. Для самых рискованных возможностей и пар «атрибут/компонент», отмеченных красным, напишите набор пользовательских историй, сценариев использования или руководство по тестированию. В Google ответственность за наиболее рискованные возможности лежит на тестировщике. Он может координировать свою работу с коллегами, использовать разные инструменты, но личная ответственность все равно на нем.
2. Внимательно изучите все то, что делалось по тестированию разработчиками и разработчиками в тестировании до вас. Как результаты повлияли на риски, выявленные с помощью GTA? Хорошо ли это тестирование было организовано с точки зрения управления рисками? Стоит ли добавить новые тесты? Тестировщику может понадобиться дописать эти тесты самому или обратиться к разработчикам. В конечном счете важно, чтобы тесты были написаны, а не кто именно их напишет.

3. Проанализируйте баги, обнаруженные у каждой пары атрибут/компонент высокого риска, и убедитесь в том, что соответствующие регрессионные тесты написаны для каждого из них. Баги имеют свойство возвращаться при изменении кода.
4. Будьте внимательнее к областям высокого риска — поинтересуйтесь механизмами восстановления и отката. Учтите возможное негативное влияние на пользователя, когда он столкнется с наихудшим сценарием. Обсудите такие ситуации с другими инженерами, проверьте реалистичность этих сценариев. К тестировщику, который часто кричит: «Волк!», вскоре перестанут прислушиваться. Громкие предупреждения о вероятных опасностях допустимы только в отношении сценариев с высоким риском, которые к тому же признаны реалистичными и уже были покрыты тестами.
5. Вовлекайте в работу как можно больше людей, заинтересованных в успешности проекта. Внутренних пользователей следует тормошить на тему обратной связи, иначе они будут просто использовать систему, игнорируя те или иные ошибки. Просите их проводить конкретные эксперименты, задавайте им вопросы типа «А как это работает на вашей машине?» или «Как бы вы использовали такую фичу?». Сотрудники Google много участвуют в тестировании, и их нужно активно направлять именно тестировать, а не просто пользоваться продуктами.
6. Если ни один из механизмов не работает, а подверженный риску компонент так и недотестиран, да еще и постоянно падает, постарайтесь добиться удаления элемента. Поздравляем! Вам выпал шанс объяснить руководству концепцию анализа рисков и подчеркнуть важность тестировщиков на проекте.

Пользовательские сценарии

Джейсон Арбон

Пользовательские истории описывают реальные или смоделированные способы, которыми пользователи используют приложение. Они описывают, чего хотят пользователи, смотрят на продукт их глазами, не учитывая архитектуру приложения и детали реализации.

Истории могут быть связаны с возможностями, но лишь поверхностно, поскольку все-таки подчинены действиям пользователя. Пользователю что-то нужно, а история описывает, как он использует приложение, чтобы это получить. Истории намеренно описаны в общем виде, без конкретных

шагов, без жестко заданных входных данных. Только то, что будет делать пользователь, и как это воспроизвести во время тестирования приложения.

Создавая пользовательскую историю, мы смотрим на продукт только через пользовательский интерфейс, мы не включаем в описание технические подробности. Тогда тестировщик будет каждый раз проходить этот путь по-разному, как и разные пользователи по-разному решают одну и ту же задачу в нашем приложении — вот в чем главная идея!

Главное в пользовательских историях — ценность продукта для пользователя. Это не тест-кейсы с их определенными вводными данными и ожидаемыми результатами. Хорошая практика — создавать отдельные учетные записи. Мы в Google часто создаем помногу тестовых учетных записей для пользователей, описанных в историях. Старые аккаунты могут быть полезны по-другому: при тестировании Google Documents мы выявили самые интересные баги как раз для старых учетных записей — при загрузке в новой версии документов, созданных в предыдущих версиях.

Мы стараемся, чтобы тестировщики, исполняя такие сценарии, менялись. Чем больше разных способов прохождения — тем лучше.

Мы не будем слишком притираться к возможностям с низкими рисками. Мы можем решить, что писать тест-кейсы для этих областей — слишком затратное занятие. Вместо этого мы можем ограничиться исследовательским тестированием или оставить на откуп краудсорс-тестированию. Чтобы управлять работой тестировщиков из внешнего сообщества, мы часто пользуемся концепцией турров — это высокоуровневые инструкции для исследовательского тестирования¹. Проще говоря, такой подход дает вашему запросу нужную конкретику. Например, попросив сообщество: «Проведите FedEx-тур для такого-то набора возможностей», — мы получим намного лучший результат, чем просто отдав приложение и понадеявшись на лучшее. Мы сразу определяем фичи, которые нужно протестировать, и даем инструкции, как это делать.

Краудсорсинг

Джеймс Уиттакер

Краудсорсинг — это новое явление в тестировании. Если тестировщиков не хватает, а их ресурсы ограничены, то краудсорс-тестирование спешит на помощь! Пользователей с разными наборами устройств и программных

¹ Подробнее о турах можно почитать у Джеймса Уиттакера в «Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design» (Addison Wesley, 2009).

конфигураций намного больше, чем тестировщиков. О таком количестве тестовых окружений нам остается только мечтать. Наверняка ведь найдутся желающие помочь нам?

Представим, что есть группа опытных пользователей, которые разбираются в тестировании и согласились нам помочь за разумную плату. Все, что им нужно, — это доступ к среде, где они могут работать с приложением, и отлаженный механизм предоставления обратной связи и баг-репортов. Для таких проектов, как наш опенсорсный Chromium, тестирование при помощи большой группы людей подходит идеально. Однако для проектов, открытых только внутри компании, это более проблематично. Нужно отбирать тестировщиков, пользующихся особым доверием.

Еще одна ключевая ценность краудсорсинга (кроме множества конфигураций) — это более широкий взгляд на приложение. Вместо того чтобы один тестировщик имитировал действия тысячи пользователей, у нас есть тысяча пользователей, работающих как тестировщики. Есть ли лучший способ найти сценарии, приводящие приложение к сбою, чем сразу выдать эти сценарии пользователям и получить обратную связь? Разнообразие и масштаб — вот в чем ценность краудсорсинга.

Людей, желающих протестировать программный продукт, в избытке, и доступны они круглосуточно. Допустим, дано: топ-1000 сайтов, задача: протестировать их в последней версии Chrome, тогда решение: 1 тестировщик = 1000 итераций или 20 тестировщиков = 50 итераций. Математика на стороне краудсорсинга.

Главный недостаток тестирования сообществом в том, что им нужно время, чтобы разобраться с приложением и понять, с какой стороны лучше подойти к тестированию. Большая часть этого времени теряется впустую из-за количества людей, но мы придумали, как с этим справляться. Для Chrome, например, мы написали туры, и внешние тестировщики следовали им при исследовательском тестировании и при выполнении пользовательских сценариев (примеры есть в приложении Б «Туры тестов для Chrome»). Туры сразу направляли тестировщиков к нужным частям приложения и давали необходимые инструкции. Фокус в том, чтобы сделать разные наборы туров и распределить их между участниками. Так мы избежали варианта «принеси то, не знаю что» и получили именно то, о чем просили.

Краудсорс-тестирование — это следующий этап развития стандартных каналов Google: канареечного канала, канала разработки, тестового канала и канала внутреннего продукта. Это наш способ привлечения ранних пользователей и людей, которым просто нравится искать баги и сообщать о них. Мы уже попробовали набирать тестировщиков внутри компании среди

наших коллег, которые любят работать со свежим продуктом, подключать к командам людей из компаний поставщиков, пользоваться услугами коммерческих компаний краудсорсинга (например, uTest). Мы даже запустили программу поощрения лучших искателей багов¹.

Итак, сила ACC-анализа в том, что мы получаем список возможностей продукта, который можно упорядочить по риску и закрепить за разными исполнителями. Тестирующие, работающие над одним проектом, могут получить разные наборы возможностей для проверки. Внутренние пользователи, «двадцатипроцентные» участники, тестирующие-подрядчики, тестирующие из сообщества, разработчики, разработчики в тестировании — все получат свои списки возможностей, и, к радости тестирующего, важные области будут покрыты с меньшим перекрытием, чем если бы мы просто раздали приложение для тестирования всем желающим.

Работа тестирующего, в отличие от работы разработчика в тестировании, с выпуском продукта не заканчивается.

Пишем тест-кейсы

Тестирующие в Google создают очень много тест-кейсов, которые определяют входные данные (от общих условий до конкретных значений) для тестирования приложения. Тест-кейсы могут строиться и на основе пользовательских историй, и прямым преобразованием возможностей продукта: работа с ними не зависит от их происхождения. В отличие от кода и автоматизации, которые управляются общей инфраструктурой, тест-кейсы до сих пор пишутся локально каждой командой. Недавно появился инструмент, который изменит ситуацию.

Чаще всего тест-кейсы хранились в электронных таблицах и документах. Команды, работающие по быстрым методологиям с короткими циклами выпуска, не очень заботятся о сохранении тест-кейсов. Как только выходит новая фича, скрипты начинают падать, и тесты приходится переписывать заново. Поэтому документа, который можно использовать, а потом с чистым сердцем удалить, — вполне достаточно. Формат такого документа не подойдет для конкретных тест-кейсов с действиями, но сгодится для описания контекста тестовой сессии. Такие тесты обычно менее регламентированы и, по сути, просто указывают, какие фичи нужно исследовать.

¹ Программа «Платим за баги» для Chrome обсуждается по адресу: <http://blog.chromium.org/2010/01/encouraging-more-chromium-security.html>

Конечно, есть команды, которые хранят тестовые процедуры и данные в довольно сложных таблицах. Некоторые даже копируют данные АСС-анализа в электронные таблицы, потому что они, видите ли, более гибкие, чем Google Test Analytics. Но такой подход требует дисциплины и целостности в команде, ведь работа одного тестировщика может ограничить работу другого. Большим командам с их текучкой нужен другой подход. Нужна структура, которая переживет любого участника команды.

Электронные таблицы все же удобнее документов, потому что в них имеются подходящие столбцы для процедур, данных и отметок о результатах. Их легко настроить под себя. Google Sites и другие разновидности онлайновых вики-систем часто используют, чтобы транслировать тестовую информацию другим заинтересованным людям. Их легко использовать совместно и редактировать.

С ростом Google у многих команд росли объемы тест-кейсов, ими нужно было лучше управлять. Некоторые документы с тест-кейсами выросли до таких размеров, что даже поиск в них стал невозможен. Нужен был новый подход. Наши тестировщики построили систему на основе нескольких коммерческих продуктов и доморощенных систем управления тест-кейсами, знакомых им по предыдущим местам работы. Систему назвали Test Scribe.

Test Scribe хранит тест-кейсы в жесткой структуре. Можно включить или исключить тест-кейс из конкретной тестовой сессии. Реализация была примитивной, и энтузиазм по поводу ее использования быстро угас. И хотя многие команды оставили что-то от нее и возились с этим несколько кварталов, мы ее все-таки похоронили. В 2010 году Джорданна Корд, старший разработчик в тестировании, написала новую программу. Это была система Google Test Case Manager (GTCM).



Рис. 3.10. Домашняя страница GTCM сосредоточена на поиске

GTCM создали, чтобы сделать процесс написания тестов проще, создать гибкий формат тегов, который можно адаптировать под любой проект, упростить поиск и повторное использование тестов. И — самое важное — интеграция GTCM с остальной инфраструктурой Google. Посмотрите на снимки экранов GTCM (рис. 3.10–3.14). На рис. 3.11 показана страница создания тест-кейса. Можно создавать произвольные разделы или метки, поэтому GTCM поддерживает любые

схемы, от классических тестов до исследовательских турнов, «огурцов»¹ и описаний пользовательских историй. Некоторые тестовые команды даже хранят фрагменты кода и данные прямо в тест-кейсах GTCM. GTCM помогает в работе любым тестовым командам и учитывает их различающиеся представления тест-кейсов.

Create Project

Project Name

Description of your project

Bug tracker Buganizer

ComponentId (Buganizer), Project Name (Issue Tracker) or Url

Deprecation Service Settings

The deprecation service monitors your tests and automatically notifies you if tests have become inactive and should be considered for archiving or deletion.

Tests marked as inactive can be reviewed within the depreciation service. You can also use search to find all tests that have been marked as potentially inactive. We recommend archiving these inactive tests to make it easier to maintain your projects. Archived tests can still be found in search using the 'isarchived' search operator but are hidden by default.

Number of days that must pass of no activity before we consider the test inactive. When we consider a test as inactive it will be added to the depreciation service for review so that you can easily decide whether to archive, delete, or leave it as is.

Disable Deprecation Service

Number of days - 30 days

Permissions

Project level permissions impact who can edit and see your project. These do not override test level permissions. This means you may grant someone access directly to a test who may not have access to the entire project.

Who can view?	Who can edit?
<input type="checkbox"/> google.com People at google.com can view. These users may view. (Valid email addresses required. Comma separated. Maximum of 25.) eng@dep.google.com	<input type="checkbox"/> google.com People at google.com can edit. These users may edit. (Valid email addresses required. Comma separated. Maximum of 25.) eng@dep.google.com

Save **Cancel**

Рис. 3.11. Создание проекта в GTCM

Submit your ideas for Q3's new features! [Copy](#)

[Bugs & Feedback](#) | [Issue Off](#) | [jarbon@google.com](#) | [Sign out](#)

Search **Results** **Create Test** **Projects** **Imports** **User Stuff**

My Saved Searches [Create New Search](#) No saved searches.

Favorite Projects No favorites.

Save as template **Save** **Cancel**

Project: Select project • No Project Selected

Templates: None

Test Name:

Manage Permissions

Use test sections to create the body of your test. For example, you might create a section for preconditions, and another for verification steps.

+ Add a section

Name: [Remove](#)

Content:

+ Add a section

Public labels (Visible by all and used in search. Useful for organizing your tests within your project)

Private labels (Only viewable by the person who added it)

Buckets [Add Bucket](#) Buckets are simply a named collection of public labels.

Save as template **Save** **Cancel**

© 2010 Google - Learn More about TCM - Join the User Group! Keyboard Shortcut Help - Shift + ? Version 22385143

Рис. 3.12. Создание теста в GTCM

1 «Огурцами» называют поведенческие тест-кейсы. Подробнее – на <http://cukes.info>

The screenshot shows the GTCM interface with a search bar at the top containing 'chrome'. Below the search bar is a table of test cases. The columns include Id, Title, Author, Creation Time, Labels, Buckets, and various status metrics. One row is highlighted in blue.

	ID	Title	Author	Creation Time	Labels	Buckets	Last Result For Test	Last Result	Last Added	Last Modified	Add Column For Results
<input type="checkbox"/>	3723271	"Chevron" after minimizing the extension separator	Srikanth@google.com	12/21/10 12:14 PST	1.0. Cross_Platform_Extensions_FromTestScribe, functional_testing_Functionality_test_Manual_P1	Buckets jchord ImportedFrom: chrome.xml	Missing	12/21/10 12:14 PDT			Add Result
<input type="checkbox"/>	3671293	"About Google Chrome" long text is truncated, if truncated, it should bring up the Chrome Info Window	venkataramana@google.com	12/21/10 12:01 PST	1.0. All_FromTestScribe, Front_end_and_backend_components_that_are_tested_manually_Frontend_Manual_C3	Buckets jchord ImportedFrom: chrome.xml	Missing	04/22/11 3:46 PDT			Add Result
<input type="checkbox"/>	3643229	"About Google Chrome" should bring up the Chrome Info Window	inderman@google.com	12/21/10 12:00 PST	1.0. All_FromTestScribe, Front_end_and_backend_components_that_are_tested_manually_Frontend_Manual_Menus_P3_Wrench_Menu	Buckets jchord ImportedFrom: chrome.xml	Missing	04/22/11 3:46 PDT			Add Result
<input type="checkbox"/>	3632290	"About Google Translate" system is	deepakg@google.com	12/21/10 12:04 PST	1.0. 37_118n_FromTestScribe_functional_testing_Functionality_tests_MAC_Manual_P1_Translation_bar	Buckets jchord ImportedFrom: chrome.xml	Missing	12/21/10 12:04 PDT			Add Result

Рис. 3.13. Просмотр тест-кейсов при поиске Chrome в GTCM

The screenshot shows a detailed view of a test case for the 'About Google Chrome' window. The test case has the following details:

- Summary:** "About Google Chrome" should bring up the Chrome Info Window
- Author:** inderman@google.com
- Project:** chrome_Basice_Prototest
- Created:** 12/21/10 12:00 PST
- Last modified:** 04/22/11 9:46 PDT by ananth@google.com

The test case includes sections for Objective, Precondition, Procedure, Verification, Notes from TestScribe Importer, and Results. The Results section indicates no results have been added. There are also sections for Bugs (no bugs have been added) and Attachments (no attachments).

Рис. 3.14. Простой тест-кейс для диалогового окна About в Chrome

Метрики, полученные от GTCM, дают представление о том, что происходит с тест-кейсами в общем. Можно понаблюдать за графиками общего количества тес-

стов и результатов тестов на рис. 3.15 и 3.16. Общее количество тестов приближается к асимптоте. Это потому, что Google закрывает старые проекты, ориентированные на ручное регрессионное тестирование, вместе с их тестами. Кроме того, GTCM в основном содержит ручные тесты, а многие команды заменяют ручные методы тестирования автоматизацией, краудсорсингом или исследовательским тестированием. Поэтому общее число тест-кейсов во внутренней базе ТСМ уменьшается, хотя покрытие при этом увеличивается. Количество проведенных тестов тоже увеличивается, так как в этой области доминируют несколько больших команд, для которых ручное тестирование обязательно (например, команда Android).

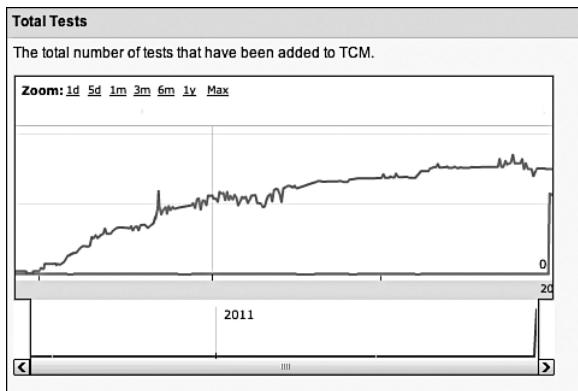


Рис. 3.15. Изменение количества тестов со временем в GTSM

Общее количество проведенных ручных тестов, как и следовало ожидать, увеличивается (рис. 3.16).

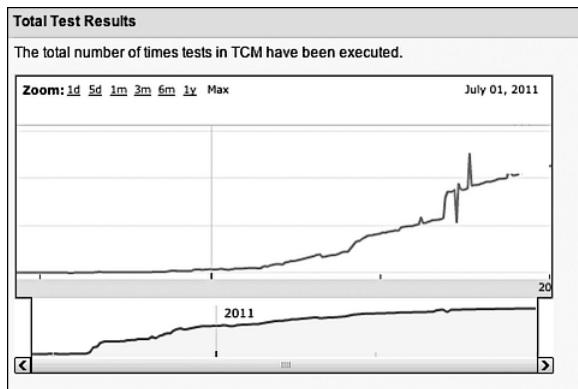


Рис. 3.16. Изменение количества результатов тестов со временем в GTSM

Посмотрим на график количества багов, показанный в GTSM (рис. 3.17). Он интересен, но еще не может показать всю картину. Google дает инженерам сво-

боду действий. Одни команды отслеживают, какие баги были найдены какими тест-кейсами, а другие не считают эти данные полезными для своего проекта. Некоторые баги создаются в системе автоматически, не все они были найдены в ходе ручного выполнения тестов.

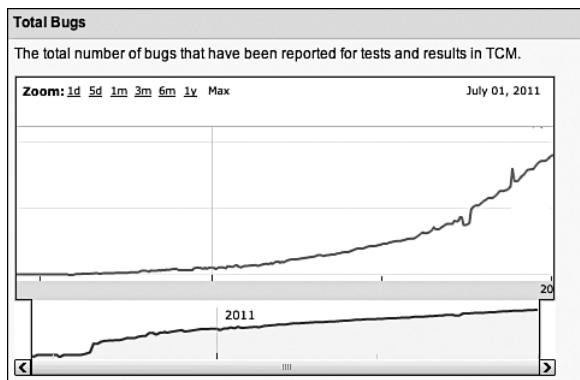


Рис. 3.17. Изменение общего количества багов, заведенных в ходе выполнения тестов GTCM, со временем

Основным требованием к GTCM с самого начала было наличие четкого и простого API. На самом деле и у системы TestScribe был API, но он базировался на SOAP, а схема аутентификации была настолько недружелюбной, что ею мало кто пользовался. Кроме того, с повышением внутреннего уровня безопасности тот режим аутентификации стал непригодным. Эти проблемы решились, когда у GTCM появился RESTful JSON API.

Команда разработки собирается скоро открыть GTCM для внешнего использования. Мы надеемся перевести эту базу данных тест-кейсов на модель открытого кода, чтобы поддерживать ее всем миром. Система GTCM проектировалась с расчетом на продолжение использования извне. Она построена на базе Google App Engine для обеспечения масштабируемости и для того, чтобы другие компании могли развернуть у себя свою копию системы. Внутренняя структура GTCM сделана так, чтобы отделить большую часть логики и пользовательских интерфейсов от Google App Engine, чтобы люди могли портировать систему. Следите за Google Testing Blog, если хотите узнать больше об этом процессе.

Интересные факты из жизни багов

Каждый тестировщик знает, что такое баги и баг-репорты. Поиск багов, сортировка багов, исправление багов, отлов регрессионных багов — основной костяк работы по контролю качества программных продуктов. Эта часть тестирования,

пожалуй, больше всего распространена в Google, но и здесь есть несколько интересных особенностей. В этом разделе мы не будем говорить о «жучках», которые заносятся в систему, чтобы отслеживать работу ее элементов. Речь пойдет только о реальных ошибках в коде. Ежедневная работа команд разработки чаще всего связана именно с ними.

Итак, сначала был баг. Баги может найти и занести в багтрекинговую систему любой сотрудник Google. Менеджеры продуктов заводят баги еще на ранних версиях продукта, когда находят расхождения со спецификацией или со своим представлением о продукте. Разработчики заводят баги, когда понимают, что случайно поломали свой код или зацепили чужой, или когда сами используют продукты Google. Баги могут прийти с полей: в ходе работы краудсорс-тестировщиков или внешнего тестирования сторонними компаниями. Коммюниити-менеджеры, которые следят за сообщениями в специальных группах, посвященных продуктам, тоже могут заводить баги. Во внутренних версиях некоторых приложений, например Google Maps, можно сообщить о баге одним кликом. Иногда баг регистрируется через API прямо из приложения.

Если процесс отслеживания багов и связанный с ним объем работы — это такая большая часть работы инженеров, понятное дело, хочется такой процесс автоматизировать. Первой попыткой подобной автоматизации в Google была система под названием BugsDB: простая база данных из нескольких таблиц и запросов, где можно было хранить и получать информацию и даже считать какую-то статистику. Системой BugDB пользовались до 2005 года, пока два предпримчивых инженера, Тед Мао¹ и Рави Гампала, не создали Buganizer.

Вместе с Buganizer мы получили новую гибкую n-уровневую иерархию компонентов взамен простой иерархии «проект → компонент → версия», которая была в BugDB (да и во всех остальных коммерческих багтрекинговых системах того времени). Стало проще учитывать и отслеживать баги, появился новый жизненный цикл рассмотрения и сопровождения багов. Упростилось отслеживание групп багов — теперь мы могли создавать и управлять целыми списками, строить диаграммы и отчеты. Разработчики реализовали полнотекстовый поиск, историю изменений, настройки «по умолчанию» при создании бага. В целом использовать новую систему стало намного удобнее, пользовательский интерфейс стал интуитивно понятным. Плюс с добавлением аутентификации повысилась безопасность.

Немного подробнее о Buganizer

Самый старый баг, зарегистрированный в Google, создан 18 мая 2001 года в 15:33 и существует до сих пор. Он называется «Test Bug», а его описание выглядит как

¹ Интервью с Тедом Мао приведено во второй главе.

«First Bug!». Смешно, что этот баг до сих пор случайно всплывает, когда разработчики привязывают исправленные баги к своим коммитам.

Самый старый из активных поныне багов был зафиксирован в марте 2009 года. В нем предлагается исследовать проблемы производительности для снижения задержки отображения рекламы с учетом локации пользователя. В последнем комментарии к этому багу написано, что вопрос может быть исследован, но это потребует архитектурной работы, а вообще — метрики задержки вполне допустимы. Это было в 2009 году.

Посмотрите на диаграммы багов в Google. Одни баги были заведены автоматически, другие вручную. Диаграмма показывает сводную информацию. Автоматизация явно лидирует, и хотя мы не выделяли данные по отдельным командам, эти диаграммы все равно довольно интересны.

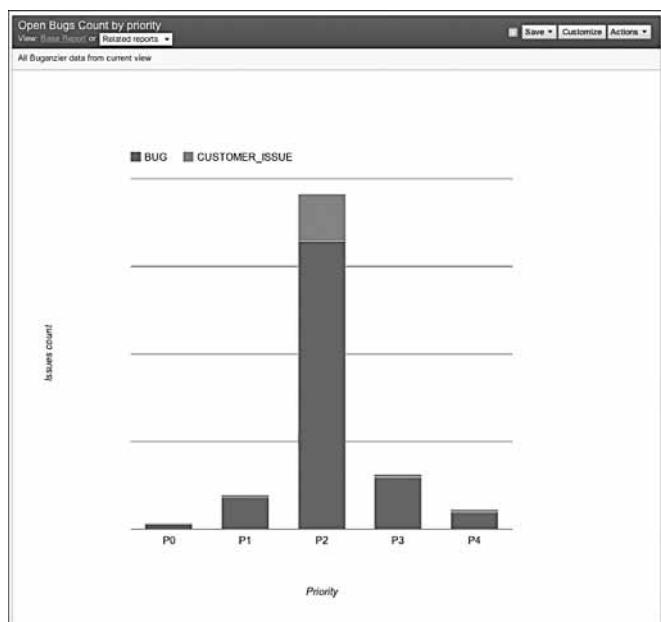


Рис. 3.18. Распределение багов в Buganizer по приоритетам

На рис. 3.18 мы видим, что есть много багов с приоритетом P2¹, значительно меньше багов P1 и совсем мало багов P0. Корреляция необязательно подтверждает причину-следствие, но вполне может быть признаком того, что инженерная методология, описанная в этой книге, действительно работает. Конечно, можно подумать, что люди просто не заводят баги с высокими приоритетами, но в ре-

1 Как и во многих багтрекинговых системах, для определения приоритета ошибки мы используем запись РХ (где Р — от слова priority, Х — целое число). Ошибки Р0 — самые сложные, ошибки Р1 менее серьезны и т. д.

альной жизни это не так. А вот баги P3 и P4 часто не заносятся в систему, так как на них редко обращают внимание.

Средний возраст багов на рис. 3.19 тоже показывает вполне ожидаемую картинку. Заметили аномалию в области багов P0? Все просто: на практике баги P0 часто трудно исправить, поскольку это серьезные просчеты в проектировании и развертывании, сложные для отладки и решения. Срок жизни остальных багов зависит от их приоритета: чем меньше важность бага, тем дольше его могут исправлять.

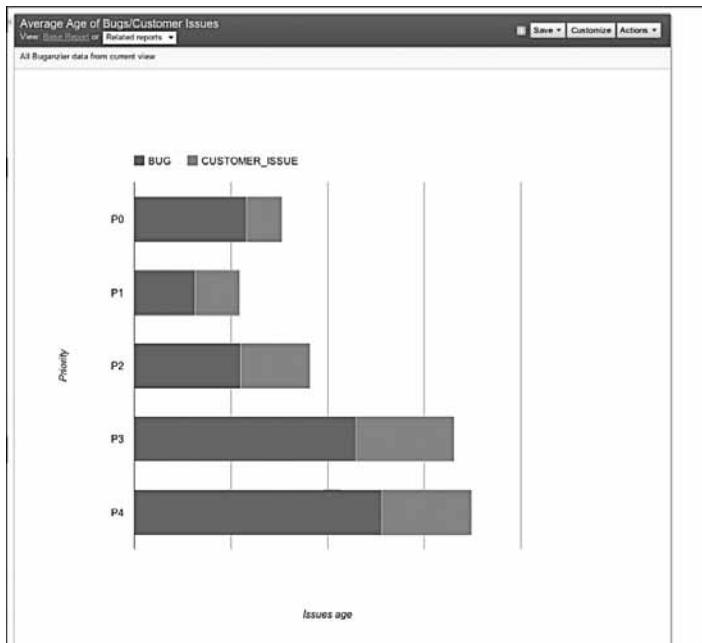


Рис. 3.19. Средний возраст багов в Buganizer

Диаграмма зависимости количества багов от времени (рис. 3.20) показывает постоянный небольшой рост. Мы не знаем точно, почему это происходит. Наверное, у нас становится больше кода. С другой стороны, количество багов растет не так быстро, как количество разработчиков и тестировщиков. Может быть, наш код и становится лучше из-за контроля качества, а может быть, мы просто их не находим.

График исправления багов на рис. 3.21 показывает нам, что команды обычно справляются с ростом багов. Многие команды просто перестают добавлять новые фичи, когда скорость появления багов начинает превышать пропускную способность команды по их исправлению. Мы рекомендуем всем перенять эту практику, а не фокусироваться только на фичах и поставленных датах завершения. Чтобы держать число багов под контролем, сосредоточьтесь на маленьких фрагментах тестируемого кода, тестируйте инкрементально и отдавайте продукт внутренним пользователям как можно раньше.

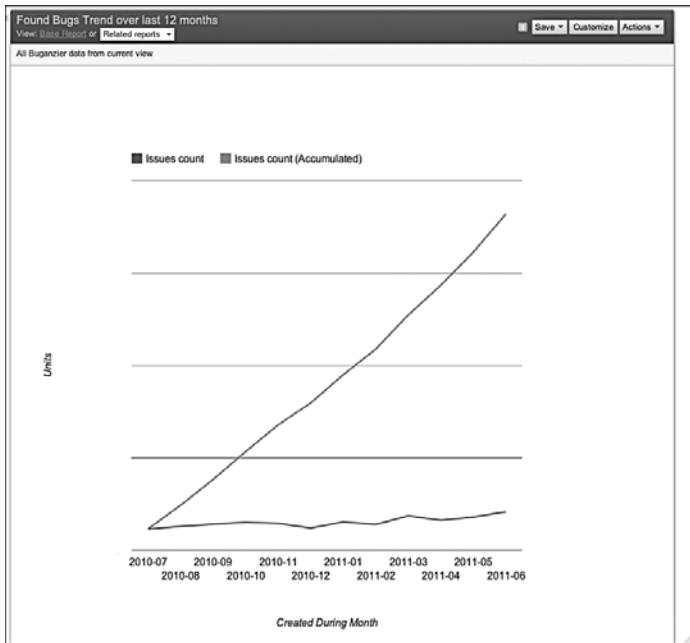


Рис. 3.20. Зависимость количества багов от времени

Как только продукты Google становятся открытыми (к примеру, Chrome и Chrome OS), нам уже не удается вести единую базу данных багов. В этих проектах используются базы данных багов, доступные извне, — Bugzilla для проблем WebKit и Issue Tracker для проблем chromium.org. Мы в Google подталкиваем наших сотрудников заводить баги из любых продуктов Google, независимо от того, работают ли они на этом проекте. Наша главная цель — сделать интернет лучше.

Issue Tracker — основной репозиторий для всех багов Chrome и Chrome OS. Эта багтрекинговая система открыта публично: информацию об исправлении багов может получить любой желающий, даже пресса. Мы иногда скрываем баги безопасности до их исправления, чтобы не раззадоривать хакеров, но в остальном доступ к базе данных открыт. Внешние пользователи могут свободно заводить баги, и для нас это ценный источник информации. На рис. 3.22 и 3.23 показан процесс и результат поиска бага, связанного с логотипом Chrome в окне About.

Как бы там ни было, а Buganizer — это самый долгоживущий и широко используемый компонент инфраструктуры тестирования в Google. В целом это типичная багтрекинговая система, но она поддерживает наш жизненный цикл отслеживания проблем в наших продуктах, от обнаружения до разрешения и создания регрессионных тест-кейсов. Чтобы система Buganizer работала быстро с нашими объемами, мы используем новейшие технологии хранения данных Google.

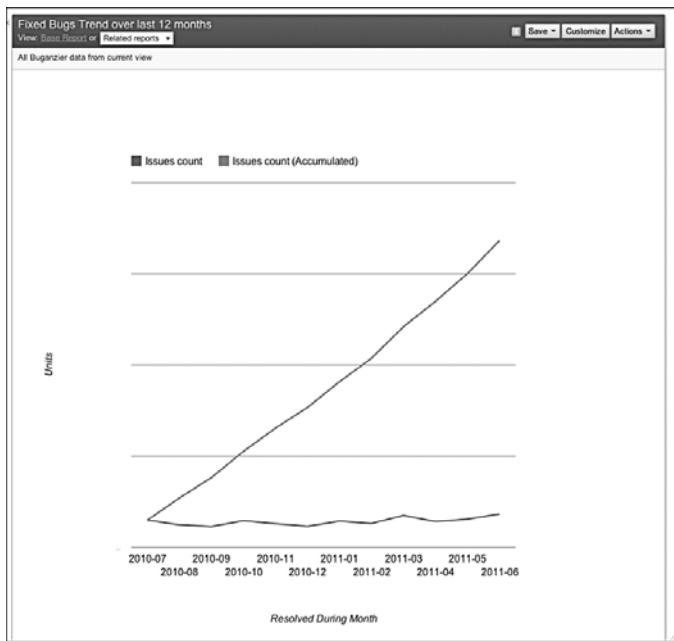


Рис. 3.21. Зависимость количества исправленных багов от времени. График похож на график количества обнаруженных багов, и это хорошо!

ID	Stars	Pri	Area	Feature	Type	Status	Summary + Labels	Modified	Owner
77354	71	1	UI	Browser, AboutBox	Regression	Assigned	REGRESSION: NEW logo reverted back to old logo in 12.0.712.0 dev	Mar 25	—
77388	3	1	UI	CloudPrint	Bug	Assigned	Mac Valgrind bot reports multiple uninitialized accesses in unit_tests (including reads, jumps and syscalls)	May 24	dmacl...@chromium.org
74604	114	2	UI	Browser	Regression	Assigned	REGRESSION: Windows painting issue while switching Chrome window with overlapped app	Jun 24	ben@chromium.org
47299	2	2	WebKit	—	Bug	Unconfirmed	When using shift-drag to move around a large SVG image the vertical scrolling is reversed MovedFrom13	Jun 2	wjmac...@chromium.org
79348	1	2	UI	—	Bug	Assigned	Chrome DMG icon needs to be updated	Jun 2	sail@chromium.org
83181	5	2	WebKit	DevTools	Bug	Assigned	Cannot interact with images in the inspector (Developer Tools)	May 19	pfeld...@chromium.org
84506	2	2	Internals	NewTabPage, Apps	Bug	Assigned	NTP app promo logo should support https URLs	Jun 27	jstri...@chromium.org
679	195	2	WebKit	—	Polish	Available	Should respect monitor DPI settings candidate bulkmove	Mar 23	—
1673	19	2	UI	—	Bug	Available	Color profile is unloaded	Jul 2009	—
5777	1	2	WebKit	—	Bug	ExternalDependency	Incorrect rendering of FF12Maps.com	Apr 15	—
7769	2	2	WebKit	—	Bug	Available	Button on New York Times is not rendered	Sep 2009	—

Рис. 3.22. Поиск в Issue Tracker

The screenshot shows the Chromium Issue Tracker interface. At the top, there's a navigation bar with links for 'Project Home', 'Downloads', 'Wiki', and 'Issues'. The 'Issues' tab is currently selected. Below the navigation is a search bar with fields for 'New issue', 'Search', 'Open issues', and a logo search field. To the right of the search bar are buttons for 'Search', 'Advanced search', and 'Search tips'. The main content area displays a single issue report. The issue summary is 'Issue 77354: REGRESSION: NEW logo reverted back to old logo in 12.0.712.0 dev'. It has a status of 'Assigned' and was reported by 'matija...@gmail.com' on March 24, 2011. The 'Cc:' field lists several email addresses. The 'What is the expected result?' section asks for the same new awesome Google Chrome logo from previous dev version. The 'What happens instead?' section notes that the new logo was changed to the old logo. There are two comments: one from 'd...@chromium.org' expressing love for the new logo, and another from the same user stating no comment was entered for the change. The bottom of the issue page shows standard labels like 'Summary', 'Status', and 'Labels'.

Рис. 3.23. Открытый баг в Issue Tracker

Чтобы завести баг, нужно заполнить поля из списка, который мы приводим ниже. Только некоторые из них обязательны. Их определения специально описаны довольно свободно, чтобы команды могли подстроить управление данными к своему рабочему процессу.

— Назначен (Assigned to, Assignee)

[Не обязательно] LDAP-аккаунт сотрудника, который должен сделать следующий шаг в обработке этого запроса. Этот человек автоматически получит сообщение при создании проблемы и каждый раз, когда кто-то изменит значение любого поля. У каждого компонента есть ответственный по умолчанию.

— Поставить в копию (CC)

[Не обязательно] Любое количество (ноль или более) LDAP-аккаунтов сотрудников, которых нужно информировать о создании или изменении проблемы. Имена задают в формате LDAP или имен почтовых ящиков без @google, поэтому можно ввести только действующие аккаунты. Элементы списка разделяются запятыми. Не нужно включать имя из поля «Назначен» при регистрации проблемы, потому что этот человек и так получит письмо.

– **Вложения (Attachments)**

[Не обязательно] Документы, прикрепляемые при создании бага (ноль или более). Принимает любой тип файлов, их количество может быть любым, но размер всего приложения не должен превышать 100 Мбайт.

– **Блокирует (Blocking)**

[Не обязательно] ID багов, решению которых препятствует этот баг. Элементы списка разделяются запятыми. Обновление такого списка автоматически обновляет поле «Зависит от» перечисленных багов.

– **Зависит от (Depends On)**

[Не обязательно] ID багов, которые должны быть исправлены до исправления этого бага. Обновление списка приводит к автоматическому обновлению поля «Блокирует» перечисленных багов. Элементы этого списка тоже разделяются запятыми.

– **Дата изменения (Changed)**

[Только для чтения] Дата и время последнего ручного изменения любого из полей проблемы.

– **Изменения (Changelists)**

[Не обязательно] Ноль или более номеров коммитов кода, связанных с решением проблемы. Указываются только коммиты, которые уже залиты в репозиторий.

– **Компонент (Component)**

[Обязательно] Компонент, содержащий баг или запрос на реализацию функциональности. Здесь нужно указывать полный путь к компоненту, причем длина пути не ограничена.

Дополнительные компоненты могут создавать только руководители проекта или разработки.

– **Дата создания (Created)**

[Только для чтения] Дата создания бага.

– **Обнаружен в (Found In)**

[Не обязательно] Используется для контроля версий; в поле указывается номер версии продукта, в которой была обнаружена проблема (например, 1.1).

– **Последнее изменение (Last modified)**

[Только для чтения] Дата последней автоматической модификации любого поля проблемы.

— Заметки (Notes)

[Не обязательно] Подробное описание проблемы и комментарии по ходу ее решения. При регистрации проблемы нужно описать шаги для воспроизведения бага или путь к экрану, который требует изменений. Чем больше информации вы здесь укажете, тем меньше вероятность того, что разработчикам, которые будут решать эту проблему, придется связываться с вами. Редактировать предыдущие записи в поле «Заметки» нельзя, даже если это вы сами их добавили; поле «Заметки» поддерживает только добавление новых значений.

— Приоритет (Priority)

[Обязательно] Определяет важность бага, где P0 — самый серьезный баг. Приоритет показывает, насколько срочно нужно исправить баг и сколько ресурсов выделить для его исправления. Например, опечатка в слове «Google» на логотипе страницы поиска будет некритичной, так как функциональность страницы от нее не страдает, но высокоприоритетной. Такие опечатки — это очень плохо. Заполнение обоих полей позволит команде исправления багов разумно распределять свое время. Обратите внимание на описание поля «Критичность» для закрепления темы.

— Кем создан (Reported by, Reporter)

[Только для чтения] Имя пользователя или сотрудника Google, первым сообщившего о баге. По умолчанию в поле заносится аккаунт человека, который завел проблему, но оно может быть изменено позднее для чествования фактического первооткрывателя.

— Резолюция (Resolution)

[Не обязательно] Последняя операция, выполняемая проверяющим. Может принимать значения «Нереально» (Not feasible), «Работает как предполагалось» (Works as intended), «Не воспроизводится» (Not repeatable), «Информация устарела» (Obsolete), «Дубликат» (Duplicate) и «Исправлено» (Fixed).

— Критичность (Severity)

[Обязательно] Критичность определяет, как сильно баг мешает использованию продукта. S0 — самые критичные баги. Мы выставляем и приоритет, и критичность каждой проблеме, чтобы разработчики могли приоритизировать важность багов при исправлении. Помните наш пример с опечаткой в слове «Google»? Ошибка высокого приоритета, но некритичная. Значения критичности можно объяснить как:

- S0 = Система непригодна к использованию
- S1 = Высокая

- S2 = Средняя
 - S3 = Низкая
 - S4 = Не влияет на систему
- **Статус (Status)**
- [Обязательно] Текущее состояние бага. Посмотрите на жизненный цикл бага на рис. 3.24, чтобы понять, как меняется значение этого поля. «Статус» бага может быть несколько видов.
- Новая (New): проблема была создана, но ответственный за ней еще не закреплен.
 - Назначена (Assigned): назначен ответственный.
 - Принята (Accepted): ответственный принял проблему.
 - Будет исправлена позднее (Fix later): ответственный решил, что проблема будет исправлена позднее.
 - Не будет исправлена (Will not fix): ответственный решил, что проблема по какой-то причине не будет исправлена.
 - Исправлена (Fixed): проблема исправлена, но исправление не проверено.
 - Назначен проверяющий (Verifier assigned): проблеме назначен проверяющий.
 - Проверена (Verified): исправление проверено проверяющим.
- **Описание (Summary)**
- [Обязательно] Общее описание проблемы. Оно должно быть как можно более понятным и однозначным; когда кто-нибудь просматривает список проблем, именно этот текст поможет решить, нужно ли дальше исследовать проблему.
- **Исправить в (Targeted To)**
- [Не обязательно] Используется для контроля версий; поле заполняется номером версии продукта, в которой проблема будет исправлена (например, 1.2).
- **Тип (Type)**
- [Обязательно] Типом проблемы может быть:
- **Баг (Bug):** из-за проблемы программа работает не так, как ожидалось.
 - **Запрос на реализацию функциональности (Feature request):** нечто, что бы вы хотели добавить в программу.

- **Проблема клиента** (Customer issue): возникла в ходе обучения или общего обсуждения.
- **Внутренняя чистка** (Internal cleanup): требует сопровождения.
- **Процесс** (Process): автоматически отслеживается через API.

- **Исправлен в (Verified In)**

[Не обязательно] Используется для контроля версий; поле заполняется номером версии продукта, в которой исправление проблемы прошло проверку (например, 1.2).

- **Проверяющий (Verifier)**

[Обязательно до разрешения проблемы] Каждой проблеме назначается один человек, который имеет право отметить ее как решенную. Этот человек должен быть назначен до того, как проблема будет готова к решению. Проверяющий — единственный человек, который может установить статус «Проверена» и закрыть баг. Проверяющий и ответственный могут быть одним человеком. Посмотрим на жизненный цикл бага.

Чем отличается обработка багов в Google от такого же процесса в других компаниях?

- **Мы открыты.** База данных багов практически полностью открыта. Любой сотрудник Google может просмотреть любой баг любого проекта.
- **Перед багом все равны.** Сообщения о багах могут создавать все, включая технических директоров и старших вице-президентов. Сотрудники Google заносят в систему баги в продуктах, которые они используют, даже если они не являются частью команды этого продукта. Внутренние пользователи часто атакуют приложения просто для того, чтобы причинить им пользу.
- **Свобода выбора.** В Google нет предписанных сверху схем работы с багами. Процесс приоритизации¹ багов зависит от команды. Иногда это индивидуальная задача, иногда она решается между тестировщиками и разработчиками в неформальном разговоре. Приоритизация может быть частью еженедельной или ежедневной планерки. Нет формальных методов, электронных очередей или Большого Брата, контролирующего работу команд. Google оставляет команде право самой организовать работу с багами.

¹ Процесс, по которому мы устанавливаем, в каком порядке рассматривать баги, и принимаем решения о том, кто и в каком порядке будет ими заниматься. Схож с процессом установления очередности оказания скорой медицинской помощи в больницах и даже обозначается тем же термином «triage».

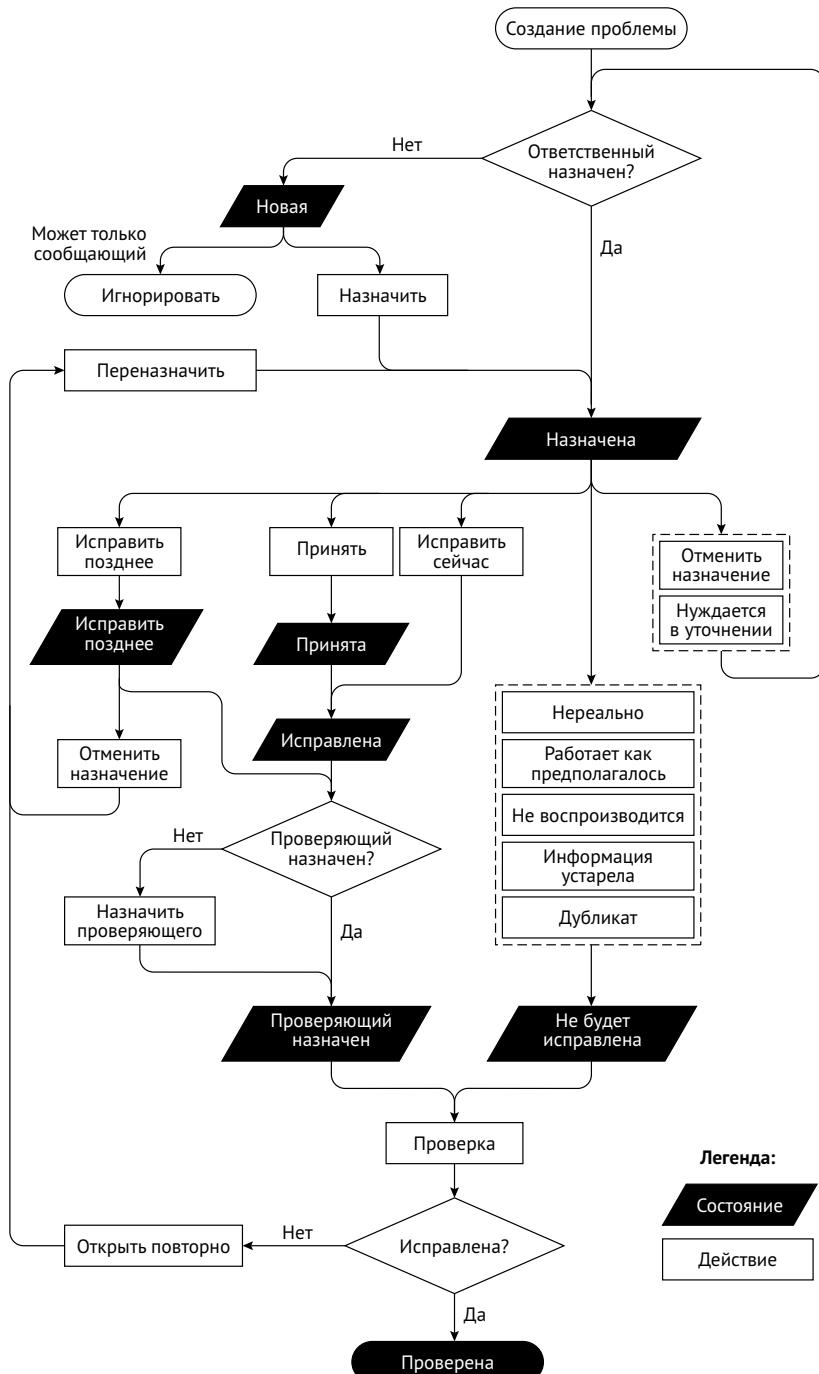


Рис. 3.24. Основной процесс устранения багов при работе с Buganizer

Проекты Google обычно находятся в одном из двух состояний: новый проект с быстрой разработкой, где поток проблем непрерывен, или проект с устоявшейся инфраструктурой и постепенными выпусками. В последнем случае объем юнит-тестирования и меры предотвращения багов сводят количество ошибок к минимуму.

Мечта создать единую панель отслеживания багов преследовала нас в Google, как идея о создании философского камня — алхимиков. Каждый год кто-нибудь пытается построить общую информационную панель для багов всех проектов. Это имело бы смысл, если бы речь шла только о данных из Buganizer. Но разные команды имеют свои представления о том, какие метрики важны для выпуска и сопровождения проектов, поэтому эти начинания с треском проваливались. Может быть, эта идея могла бы сработать в более однородной компании, но в Google такие благие намерения разбиваются о великое разнообразие проектов и практик.

Команда Google Feedback¹ (<http://www.google.com/tools/feedback/intl/en/index.html>) запустила новый подход к регистрации багов в масштабе всего Google. С помощью Google Feedback конечные пользователи сообщают о багах в некоторых сервисах Google. Идея в том, что внешние пользователи не знают, какие баги уже были заведены или исправлены, но мы хотим быстро и легко получать обратную связь от них. Соответственно тестировщики и разработчики Google сделали так, чтобы сообщение о багах осуществлялось по простому принципу «навести и кликнуть». Инженеры команды очень постарались, чтобы пользователи при отправке сообщений о багах в Google могли скрыть те части страницы, где видна их конфиденциальная информация (пример на рис. 3.25).

Команда Google Feedback здорово потрудилась над тем, чтобы наши базы данных не затопило бушующим потоком багов. Количество дубликатов могло бы просто застопорить процесс приоритизации. Они внедрили кластерные алгоритмы, чтобы автоматически группировать дубликаты и определять самые частые проблемы. За неделю после выпуска новой версии могли поступить десятки тысяч сообщений, которые в итоге сводились всего к десятку основных проблем. Это сэкономило уйму времени, и Google смог услышать своих пользователей без индивидуальной обработки каждого сообщения. Команда сейчас тестирует работу этой системы на нескольких продуктах, но собирается расширить ее на все ресурсы Google.

¹ Интервью с Брэдом Грином, техническим менеджером Google Feedback, можно прочитать в главе 4.

Google feedback

Help improve Google's products

Sending your feedback is easy using Google Feedback. Just click the feedback link, mark up the page, enter a description, and send your feedback straight to Google.

We require either of the following browsers:

- Google Chrome
- Firefox 3.5 or newer
- Internet Explorer 7 or newer, provided that it has Adobe Flash 9 or newer

Mark
Highlight or black out parts of the page

Comment
Explain the problem or make a suggestion

Send
With the click of a button, you're done!

[How to use Google Feedback »](#)

©2011 Google - [Privacy Policy](#) - [Terms of Service](#)

Рис. 3.25. Google Feedback с фильтрацией конфиденциальной информации

Жизненный путь бага

Джеймс Уиттакер

Баги напоминают детей, излишне опекаемых родителями. Они получают много внимания. Они рождаются в тиши IDE на машине разработчика, но как только они выходят в большой мир, их жизнь наполняют фанфары.

Если тестировщик нашел баг, дальше его путь выглядит так: тестировщик выделяет минуту-другую, чтобы насладиться находкой. Серьезно, это важный этап, его нельзя пропускать. Это не только дает нам возможность порадоваться плодам наших трудов, но и помогает понять неочевидные нюансы и обстоятельства возникновения бага. Лежит ли он на пути, по которому ходят пользователи? Насколько такой путь вероятен? Есть ли другие пути, которые приведут к нему? Есть ли побочные эффекты, которые могут повлиять на данные или другие приложения? Это может повысить критичность. Какие последствия для конфиденциальности, безопасности, производительности или доступности? Тестировщик чувствует себя как родитель, который, услышав легкий кашель своего ребенка, с ходу представляет самые опасные болезни.

Как и родитель, который звонит другу или родственнику, чтобы обсудить тревожный кашель, тестировщик должен пригласить коллегу и продемонстрировать ему больного. Поинтересоваться его мнением, поделиться своим мнением о баге — его критичности, приоритете и побочных эффектах. Ясность часто рождается в подобных обсуждениях. Родитель избавляется от ненужного вызова неотложки, а тестировщик обнаруживает, что баг, который он относил к категории P0, на самом деле тривиален и панику можно отставить.

А теперь пора заняться составлением баг-репорта. Как родитель, который тянеться за градусником, тестировщик расчехляет свои инструменты. Родитель хочет точно диагностировать болезнь. Мать хочет убедить доктора в серьезности болезни ее ребенка. Тестировщик тоже хочет повлиять на критичность, но что еще важнее — он хочет, чтобы баг легко было исправить. Снимки экрана, записи нажатий клавиш, данные трассировки стека, дампы DOM — все это пригодится для документирования бага. Чем больше информации будет у разработчика, тем легче будет найти причину и тем выше вероятность ее исправления.

Отчет о баге начинается с рассылки по электронной почте всем заинтересованным лицам, и для исправления создается список изменений кода. Он ставится в очередь на рецензирование, а после утверждения отправляется в очередь на сборку. Это лекарство от багов, и подобно тому, как родители наблюдают за реакцией своего ребенка на антибиотики, тестировщик получает по электронной почте оповещения о готовности новой тестовой сборки. Он устанавливает сборку и заново запускает проблемный тест-кейс.

Этот тест становится частью регрессионного пакета приложения. Его стремятся автоматизировать, чтобы проще было ловить регрессионные баги. Или, как минимум, создают ручной тест-кейс и заносят в систему управления тестами. Так формируется иммунитет системы к будущим заражениям — подобно тому, как ребенок получает иммунитет к тому виду бактерий, от которого он когда-то заболел.

Как мы нанимаем инженеров по тестированию

Мы в Google серьезно подходим к найму людей. У большинства наших инженеров высшее образование в области компьютерных технологий. В очень немногих учебных заведениях учат тестированию, поэтому до сих пор есть проблемы с наймом хороших тестировщиков. Сложно найти нужное сочетание навыков программирования и тестирования.

Найти специалиста на роль инженера по тестированию особенно тяжело, потому что лучшие из них не просто технари. Тестировщики в Google разрушают привычное представление об инженерах, на котором основан процесс найма разработчиков и разработчиков в тестировании. Откровенно говоря, мы не всегда все делали правильно и приносим извинения всем тестировщикам, которые пострадали от наших опытов совершенствования процесса собеседования. Инженеры по тестированию — редкие птицы. Они технически подкованы, заботятся об интересах пользователя, понимают продукт на всех уровнях от системной архитектуры до внешнего представления. Они неунывающие и общительные, и, что самое важное, изобретательные и умеют работать в условиях неопределенности. Только чудом можно объяснить, как Google (да и любая другая компания) умудряется поймать таких редких особей.

Часто забывается, что тестирование — это по сути проверка. Делает ли приложение то, что должно делать? Большая часть работы по тестированию сводится к планированию и выполнению проверок. Да, приложение может падать в процессе, но добиться сбоя — не цель тестировщика. Давить на программу, пока она не сломается, интересно, но куда интереснее давить на нее понемногу, снова и снова, имитируя ее реальную эксплуатацию пользователем. А как приятно убедиться в том, что она *не сломается* при таких условиях! Мы ищем именно такой позитивный взгляд на тестирование, когда проводим собеседование.

Мы опробовали несколько стилей собеседования для инженеров по тестированию.

- **Сначала мы собеседовали их так же, как разработчиков в тестировании.** Если мы находили умного и изобретательного кандидата, который недостаточно хорошо для нас умел программировать, мы рассматривали его на роль тестировщика. Такой подход привел к проблемам, возникала неформальная иерархия в команде. Что еще хуже, таким способом мы бы никогда не добрались до кандидатов, которые ориентированы больше на пользовательские аспекты — насколько удобно использовать продукт или как он решает задачи пользователей. Таких ребят нельзя было пропускать.
- **Потом мы снизили требования к умению программировать.** Мы подумали, что если сосредоточиться только на пользовательском и функциональном тестировании, круг потенциальных тестировщиков увеличится. Если кандидат не может написать код решения головоломки «Судоку» или оптимизировать алгоритм быстрой сортировки, это не значит, что из него не вырастет тестировщик. Это мог быть шанс привлечь в компанию больше тестировщиков, но внутри компании им было бы непросто вырасти. Культура программирования в Google настолько сильна и большинство инженеров настолько технически круты, что карьера у такого высокоуровневого тестировщика просто бы не сложилась.

- **Мы пришли к смешанному подходу.** Сегодня мы на собеседованиях ищем общие знания в области компьютерных технологий и технические навыки, при этом требуем от кандидатов хорошо разбираться в тестировании. Умение программировать необходимо, но только в рамках обязанностей тестировщиков, которые мы описывали выше: модифицировать чужой код и уметь написать скрипт для сквозного пользовательского сценария. Плюс к этому мы ждем от кандидатов сильных навыков общения, умения мыслить системно и эмпатии к пользователю. Только с таким миксом навыков можно эффективно работать и расти в Google.

Найти хороших тестировщиков трудно, потому что они должны быть хорошими во многих вещах одновременно. Не хватает хотя бы одной — собеседование окончено. Тестировщик и есть тот самый «и швец, и жнец». Самые крутые из них часто принимают окончательное решение о том, выпускать или не выпускать продукт. Если бы мы не относились к качеству этих людей так серьезно, у нас бы давно были неприятности.

В результате инженер по тестированию, которого самого протестирували по полной программе, может адаптироваться почти к любому продукту в любой роли. Начав с создания инструментов, привлечения пользователей, координации работы с другими командами и т. д., тестировщик часто начинает руководить разработчиками в тестировании, потому что шире смотрит на задачи и видит больше проблем и рисков.

Приложения становились все более сложными, пользовательские интерфейсы становились сложнее, чем google.com, поэтому армия тестировщиков в Googleросла. Пользователей становилось больше, и наши продукты стали занимать важное место в их жизни. Поэтому роль тестировщиков стала более значительной в культуре Google.

Как отличить тестировщика от разработчика в тестировании

Джейсон Арбон

Роли разработчика в тестировании и инженера по тестированию взаимосвязаны, но между ними есть фундаментальные различия. Я был на обеих позициях и управлял обеими ролями. Взгляните на списки, которые я привожу ниже, и выберите, какое описание больше подходит вам, — может, вам стоит сменить профессию.

Вам больше подходит роль разработчика в тестировании, если вы:

- Можете взять спецификацию и запрограммировать надежное и эффективное решение с чистого листа.
- Программируя, вы вините себя, что не написали все юнит-тесты, которые могли бы. Вы обнаруживаете себя за раздумьями, как бы сгенерировать тестовый код, чтобы не писать юнит-тесты вручную.
- Вы думаете, что конечный пользователь — это тот, кто вызывает функцию API.
- Вас раздражает плохо написанная документация API, но вы иногда забываете, зачем этот API вообще нужен.
- Вы с увлечением болтаете с другими людьми об оптимизации кода или о выявлении ошибок совместного доступа потоков.
- Вы предпочитаете общаться с другими человеческими существами через мессенджер или в комментариях к коммитам.
- Вы предпочитаете командную строку графическому интерфейсу и редко прикасаетесь к мыши.
- Вы мечтаете о том, как ваш код выполняется на тысячах машин, тестируя алгоритмы, подтверждая правильность их работы только через количество тактов процессора и сетевых пакетов.
- Вы никогда не замечали и не меняли обои своего рабочего стола.
- Вы переживаете, когда видите предупреждения компилятора.
- Когда вас просят протестировать продукт, вы открываете исходный код и начинаете думать, где прикрутить заглушку.
- Успех для вас — это построить прекрасный низкоуровневый фреймворк юнит-тестирования, который используют все и который запускается на тестовом сервере миллионы раз в день.
- Когда вас спрашивают, готов ли продукт к выпуску, вы можете просто ответить: «Все тесты прошли».

Вам больше подходит роль инженера по тестированию, если вы:

- Можете взять существующий код, поискать в нем ошибки и немедленно понять, где могут быть сбои, но вас не особенно привлекает идея писать код с нуля или даже менять его.
- Вы лучше почитаете Slashdot или News.com, чем будете весь день читать чужой код.

- Если вы читаете недописанную спецификацию, вы сами заполняете все пропуски в документе.
- Вы мечтаете поработать над продуктом, который сильно повлияет на жизнь людей и будет у всех на слуху.
- Вам становится плохо при виде пользовательских интерфейсов некоторых веб-сайтов. Вы не понимаете, как с ними вообще кто-то работает.
- Визуализация данных вызывает у вас радостное возбуждение.
- Вы ловите себя на желании пообщаться с людьми из *реального мира*.
- Вы не понимаете, почему вам нужно ввести «*i*», чтобы набрать текст в одном известном текстовом редакторе¹.
- Успех для вас — помочь реализоваться идеям других инженеров, а потом испытать эти идеи в боевых условиях.
- Когда вас спрашивают, готов ли продукт к выпуску, вы можете сказать: «Я думаю, да».

Тестировщики должны знать, кто они на самом деле. Про них часто думают, что это разработчики в тестировании, которые просто меньше программируют. На самом деле тестировщики видят то, что никогда не видят человек, копающийся целыми днями в коде. Разработчики в тестировании должны понять, что они в первую очередь не тестировщики, перестать выискивать проблемы пользовательского интерфейса, размышлять о системе в целом или о продуктах конкурентов. Вместо этого им нужно сосредоточиться на качественных, пригодных для тестирования и повторного использования модулях и создании превосходной автоматизации.

Обе роли очень важны, чтобы создать великолепный продукт.

Собеседование с инженерами по тестированию

Ура, мы обнаружили необходимую комбинацию навыков и немедленно зовем кандидата на интервью. Нас часто спрашивают, как мы собеседуем инженеров по тестированию, причем это самый частый вопрос, который нам задают буквально

¹ Речь идет о текстовом редакторе vim (<http://ru.wikipedia.org/wiki/Vim>), в котором нужно специально входить в режим ввода текста. — Примеч. перев.

взде. Полный список вопросов мы приводить не будем, но покажем несколько примеров, чтобы проиллюстрировать наш подход. Учтите, что после публикации эти вопросы уже неактуальны!

Для начала мы проверяем способности кандидата к тестированию. Наша цель — определить, выходит ли кандидат за рамки обычной сообразительности и изобретательности и есть ли у него природный дар к тестированию. Мы ищем врожденную любознательность к устройству систем, к комбинациям переменных и конфигураций, которые можно и интересно было бы протестировать. Нам нужен человек, который поймет, как система должна работать, и сможет это объяснить простыми словами. И еще у него должен быть сильный характер.

Мы даем тестовую задачу, зависящую от многих входных факторов и условий среды, а потом просим кандидата перечислить самые интересные из них. Например, мы просим кандидата протестировать веб-страницу (рис. 3.26) с одним полем ввода и кнопкой, при нажатии которой подсчитывается количество букв «A» в текстовой строке. Вопрос: составьте список входных строк, которые вы хотите протестировать.



Рис. 3.26. Простой пользовательский интерфейс для тестового вопроса

Некоторые кандидаты бросаются с места в карьер и начинают строчить тест-кейсы. (Звучит тревожная музыка.) Скорее всего, они недостаточно подумали над задачей. Мы воспринимаем стремление к количеству, а не к качеству, как знак опасности. Мы пришли к этому с опытом. О кандидате можно многое узнать еще до того, как он справится с задачей, просто посмотрев на то, как он ее решает.

Лучший кандидат — тот, который начинает задавать уточняющие вопросы: прописные или строчные буквы? А только ли английские? А текст стирается после вычисления ответа? Как насчет повторного нажатия кнопок? И так далее.

Итак, когда все вопросы прояснились, кандидаты начинают писать тест-кейсы. Важно понять, есть ли в их безумии своя система. Пытаются ли они просто сломать программу или хотят еще проверить, что она работает? Они осознают, когда делают первое, а когда второе? Начинают ли они с очевидных и простых вещей, чтобы

найти самые важные баги как можно быстрее? Могут ли они четко изложить свой тестовый план/данные? Случайный порядок строк на доске не свидетельствует о ясности мысли, а тест-планы таких кандидатов, скорее всего, будут небрежными, если они вообще позаботятся о них. Типичный список может быть примерно такой:

- «banana»: 3 (реально существующее слово)
- «A» и «а»: 1 (простой допустимый случай с положительным результатом).
- «»: 0 (простой допустимый случай с нулевым результатом).
- null: 0 (простой ошибочный случай).
- «AA» и «aa»: 2 (случай, в котором количество символов > 1 , а строка состоит из одних «A»).
- «b»: 0 (простой непустой допустимый случай с негативным результатом).
- «aba»: 2 (искомый символ в начале и конце строки для выявления багов смещения на 1 в цикле).
- «bab»: 1 (искомый символ в середине строки).
- пробелы/табуляции/и т. д.: N (символы-пропуски чередуются с N символами «A»).
- длинная строка без «A»: 0
- длинная строка с «A»: N, где N равно количеству «A».
- X\nX в строке: N, где N равно количеству «A» (символы форматирования).
- {java/C/HTML/JavaScript}: N, где N равно количеству «A» (символы исполняемого кода, бага или случайная интерпретация кода).

Если кандидат пропускает какие-то из перечисленных тестов, снова звучит тревожная музыка.

Лучшие кандидаты не ограничиваются конкретным выбором входных данных задачи и переходят на более глубокие уровни тестирования. Они могут:

- Исследовать оформление, цветовую палитру и контрастность: соответствуют ли они оформлению других взаимосвязанных приложений? Доступны ли они для пользователей с дефектами зрения и т. д.?
- Побеспокоиться о том, что текстовое поле слишком короткое, и предложить увеличить его, чтобы стало удобнее вводить длинные строки.
- Поинтересоваться возможностью запуска нескольких экземпляров этого приложения на одном сервере. Нет ли опасности пересечения данных разных пользователей?

- Спросить, сохраняются ли данные? В них могут быть адреса или другая личная информация.
- Предложить автоматизировать тест с реальными данными, например загрузить страницы из словаря или фрагменты текста из книги.
- Спросить, достаточно ли быстро это работает? А как будет работать под нагрузкой?
- Уточнить, как пользователь будет попадать на эту страницу и легко ли ее найти?
- Ввести код HTML и JavaScript. Не сломает ли это отображение страницы?
- Спросить, должны учитываться символы «A» только верхнего или только нижнего регистра либо обоих?
- Попробовать копировать и вставлять строки.

А некоторые кандидаты идут еще дальше, их не удержать рамками поставленной задачи. Вот они — опытные и ценные инженерные кадры. Они могут:

- Понять, что если данные передаются серверу в HTTP GET-запросе с URL-кодированием, строка может быть обрезана в процессе передачи по интернету. Поэтому нет никакой гарантии, что поддерживается любая длина URL-адресов.
- Предложить параметризацию приложения. Почему мы считаем только «A»?
- Подумать о возможности подсчета «A» из других языков (например, со знаками ангстрема или умляута).
- Подумать о возможности интернационализации этого приложения.
- Подумать о написании скриптов или вручную составить выборку строк определенной длины (допустим, степеней двойки), чтобы найти предел длины строки и убедиться в том, что значения внутри пределов обрабатываются правильно.
- Учесть возможную реализацию и ее код. Возможно, в реализации используются счетчик для перебора символов строки и переменная-накопитель для хранения количества обнаруженных символов «A». Тогда интересно было бы менять как общее количество «A», так и длину строки в районе граничных значений.
- Задать вопросы: «Могут ли метод HTTP POST и параметры подвергнуться хакерской атаке? Нет ли здесь проблем с безопасностью?»

- Написать скрипт для генерации данных и проверки результатов, чтобы попробовать побольше интересных комбинаций свойств строки: длина, количество «A» и т. д.

Обратив внимание на длины строк, которые кандидат использует в своих тестах, можно понять, как он будет работать. Если он ограничивается понятием «длинная строка», а такое бывает часто, снова включается тревожная музика. Продвинутые кандидаты обычно просят спецификацию строки, а затем прикидывают граничные тесты для нужных значений. Например, если максимальная длина строки 1000 символов, они попробуют 999, 1000 и 1001. Лучшие кандидаты возьмут еще и значение 2^{32} , и несколько промежуточных хитрых значений, например степени 2 и 10. Кандидаты должны понимать, какие значения важны для системы, а не просто брать случайные числа. Для этого им нужно хорошо разбираться в основных алгоритмах, языке, оборудовании и среде выполнения программ, потому что именно здесь водится большинство багов. Они попробуют варианты длины строки с учетом возможной реализации и подумают о возможности смещения счетчиков и указателей на единицу. Самые лучшие кандидаты понимают, что система может запоминать состояние и что в тестах нужно учесть значения, введенные раньше. Попробовать ввести одну и ту же строку несколько раз или проверить нулевую длину сразу после тысячи символов — важные сценарии, которые хорошо бы учесть.

Еще одно важное качество, которое мы ищем на собеседованиях, — это способность тестировщика управляться с неоднозначностью и замечать бессмысленную постановку задачи. Когда кандидаты задают уточняющие вопросы, мы часто меняем требования или описываем поведение, которое не имеет смысла. То, как кандидаты справляются с этим, показывает нам, как хорошо они будут работать. При нашей скорости выпусков спецификации часто открыты для интерпретаций и изменений. Если кандидат замечает, что ограничение длины строки в 5 символов выглядит странно и, скорее всего, будет раздражать пользователей, это означает, что он подумал о пользователе. Если кандидат слепо принимает странное условие и идет дальше, то, вероятно, он сделает то же самое в ходе работы, и бессмысленное поведение успешно пройдет проверку. Кандидаты, которые ставят под сомнение нелогичные условия, обычно творят на работе чудеса. Конечно, если им удается сделать это дипломатично.

В последней части собеседования на роль инженера по тестированию мы оцениваем его параметр «гугловости». Нам нужны особенные люди, которые влюблены в свою работу, любознательны, способные не просто выполнять, что им скажут, а исследовать все варианты. Мы хотим работать с теми, кто выходит за рамки обязанностей, экспериментирует. Нам нужны люди, открытые окружающему миру и тесно связанные с сообществом компьютерных технологий. Например, ребята, которые сообщают о багах в опенсорс-проектах, или те, кто выкладывает

свою работу, чтобы другие люди могли повторно ее использовать, имеют высокий уровень «гугловости». Мы хотим нанимать людей, с которыми приятно работать, которые хорошо ладят с другими и привносят в нашу культуру что-то свое. Если вы можете нас чему-то научить — добро пожаловать к нам! Помните наш девиз? Он звучит: «Не делай зла». Нам нужен тот, кто укажет на зло, если увидит.

Мы отлично знаем, что интервью в крупных технологических компаниях вызывают мандраж. Мы помним, как сами переживали это. Многие не проходят собеседование с первого раза, но зато получают шанс подготовиться и набраться опыта к следующему. Давайте проясним: мы не хотим быть жесткими. Наша задача — отобрать людей, которые внесут свой вклад в компанию и, став инженерами по тестированию в Google, продолжат расти у нас. Результат должен быть хорошим и для компании, и для кандидата. Google, как и большинство компаний, которые, несмотря на большое число людей, хотят сохранить эффект маленькой уютной компании, предпочитает перестраховаться с наймом сотрудников. Мы хотим, чтобы Google оставался местом, в котором нам хотелось бы проработать еще много лет. А для этого нам нужна хорошая компания!

Управление тестированием в Google

Мы иногда шутим, что умеем управлять только профессиональными, целеустремленными и самостоятельными инженерами. Если серьезно, то руководить тестировщиками в Google непросто. Как это делать правильно? Воодушевлять, нельзя командовать. Поддерживать слаженную работу. Давать людям свободу действий, вдохновлять на эксперименты, доверять им самим принимать решения. Очень легко написать — и очень сложно выполнить.

Управлять тестировщиками в Google — задача, очень сильно отличающаяся от управления обычным отделом тестирования. Причин тому несколько: у нас намного меньше тестировщиков, мы берем на работу только очень компетентных специалистов, ценим и поддерживаем разнообразие и самостоятельность в работе. В нашем понимании управление тестированием — это больше про «вдохновлять», чем про «руководить». Это больше про стратегию, чем про ежедневное руководство к действию. Наш подход делает управление тестированием более сложным и даже непредсказуемым, если сравнивать с нашими прошлыми местами работы. Чтобы управлять тестированием в Google, нужно уметь быть лидером и смотреть далеко вперед, уметь общаться и договариваться, быть подкованным технически, уметь проводить собеседования и разбираться в людях, нанимать людей и потом регулярно оценивать их работу.

Обычно в Google чрезмерная формализация управления вызывает напряженность. Перед директорами по тестированию, менеджерами и руководством Google

стоит сложная задача — дать инженерам столько свободы, чтобы не ограничить их работу лишними условностями, но и не позволять им расслабляться и плевать в потолок. Google не жалеет ресурсов на крупномасштабные, стратегические решения. Наши менеджеры помогают избежать разработки дубликатов тестовых фреймворков и чрезмерных вложений в малые тесты, при этом они помогают собирать команды для масштабных проектов. Без такого присмотра внутренние проекты тестирования часто не выживают, если остаются всего лишь проектом одного инженера или не выходят за рамки «двадцатипроцентного» времени.

Управление «пиратским кораблем» для чайников

Джеймс Арбон

«Пиратский корабль» — метафора, которую мы используем, когда говорим про управление командами тестирования в Google. Наше тестирование — это мир, в котором инженеры по природе своей постоянно задают вопросы, требуют убедительных данных и постоянно оценивают решения своего руководства. Помните, что мы набираем в свою команду самостоятельных и инициативных людей, которые сами могут позаботиться и о себе, и о своей работе. Но и ими же нужно и управлять.

Я думаю, что это примерно так же, как управляет капитан своим пиратским кораблем. Дело в том, что капитан не может держать всю команду в узде с помощью грубой силы или страха, так как подчиненных много и все они вооружены до зубов отточенными техническими талантами и другими предложениями работы. Он не может управлять только одним золотом — пираты часто получают больше, чем им нужно для существования. Что действительно вдохновляет пиратов — это их образ жизни и волнение, испытываемое при виде очередной добычи. Всегда есть реальная угроза бунта, ведь структура Google очень динамична. Инженеров даже поощряют часто менять команды. Если корабль не нашел достаточно сокровищ или на нем невесело плыть, наши пираты-инженеры сходят с него в ближайшем порту и не возвращаются, когда приходит время поднимать паруса.

Руководить инженерами означает самому быть пиратом, но знать чуть больше о том, что лежит за горизонтом, какие корабли проплывают поблизости и какие сокровища томятся в их трюмах. Другими словами, это означает управлять с помощью своего технического видения, обещать волнующие технические приключения и интересные порты для стоянок. Технические руководители в Google всегда должны держать нос по ветру.

В Google есть несколько типов руководителей и менеджеров.

- **Ведущий инженер** (Tech Lead). Эти ребята появляются в больших командах, где много разработчиков в тестировании и тестировщиков работают над большим проектом с общими техническими проблемами и инфраструктурой. Они встречаются в командах, которые занимаются инфраструктурами, не зависящими от продукта. Обычно ведущие инженеры не управляют людьми. К ним инженеры обращаются с техническими проблемами или вопросами по тестированию. Часто это неформальная роль, которая выделилась естественным образом в процессе работы. Ведущий инженер всегда сосредоточен только на одном проекте.
- **Технический менеджер** (TLM, Tech Lead Manager). Это удивительное создание рождается, когда человек, к которому обращаются по техническим вопросам, еще и официально руководит командой инженеров проекта. Технические менеджеры очень уважаемы и влиятельны. Обычно они тоже работают только над одним важным проектом.
- **Тест-менеджер** (Test Engineering Manager). Те, кто координирует инженерную работу нескольких команд, называются тест-менеджерами. Почти все они выросли из рядовых сотрудников. Можно сказать, что эта роль в Google — аналог принятой в индустрии роли тест-менеджера, но с более широким полем действий, типичным для директоров в других компаниях. В их полномочиях выделять и забирать ресурсы тестирования на проектах. Обычно у них в подчинении от 12 до 35 человек. Тест-менеджеры распределяют людей между командами, регулируют совместное использование инструментов и процессов и рулят процессом поиска, собеседований и найма людей.
- **Директор по тестированию** (Test Director). У нас несколько таких директоров. Они немного координируют работу нескольких тест-менеджеров, а по большей части работают над общей перспективой тестирования, стратегическим управлением и иногда — преобразованиями в технической инфраструктуре и методологиях тестирования. Все-таки основное направление их работы — следить, как влияет качество тестирования на бизнес: анализ затрат, анализ эффективности и т. д. Их работа развернута наружу — показывать результаты другим. У директоров по тестированию обычно от 40 до 70 подчиненных. Каждый из них закреплен за отдельным направлением: Client, Apps, Ads и т. д.
- **Старший директор по тестированию** (Senior Test Director). Такой человек у нас один — Патрик Коупленд. Он отвечает перед высшим руководством компании за единообразие должностных инструкций, найм, внешние взаимодействия и общую стратегию тестирования в Google. Это его работа — распространять передовые практики, создавать и продвигать новые

начинания, например инфраструктуру глобальной сборки или тестирования. Статический анализ и задачи по тестированию, которые касаются всех продуктов Google, пользовательские проблемы и база тестового кода — все это лежит в ответственности старшего директора по тестированию.

Большинство тех, кто работает в тестировании в Google, особенно директора и даже сам Патрик Коупленд, участвуют в поиске и наборе людей. Есть несколько подводных камней, на которые мы натыкаемся во время собеседований. Например, большинство кандидатов отлично знают компанию, ее основные технологии и то, что это отличное место для работы. Инженеры очень осторожны на собеседовании: они опасаются показать свое волнение и не пройти отбор. Хорошие кандидаты и так работают в хороших условиях, и их беспокоит сильная конкуренция внутри Google. Часто можно развеять все опасения, если интервьюер представится и расскажет немного о себе. Инженеры в Google, безусловно, профессиональные и воодушевленные. И все же большая часть того, что заставляет их работу выглядеть такой интересной и даже героической, — это просто грамотное использование коллективного разума сообщества инженеров-единомышленников и технических мощностей Google.

У нас есть и процесс внутреннего подбора людей. Инженеров поощряют менять проекты, поэтому движение между командами есть всегда. Если команде нужны люди, ей нужно рассказывать всем о том, почему у них круто работать. Большинство внутренних перемещений — результат личного общения, когда инженеры обсуждают интересные проекты, технические проблемы и общую атмосферу в команде. Иногда проводятся полуформальные встречи, на которых команды, желающие привлечь инженеров, показывают, над чем они работают. Но по большей части внутренний набор людей в команды — процесс очень естественный и гармоничный. Люди просто *могут* перейти туда, где они хотят работать и где, как им кажется, их работа принесет наибольшую пользу.

Давайте отметим основные аспекты работы руководителей тестирования.

- **Технический момент.** Тест-менеджеры и особенно ведущие инженеры должны быть технически сильными. Им придется писать прототипы, проводить код-ревью. Они всегда должны стремиться узнать продукт и пользователей лучше, чем все остальные участники команды.
- **Переговоры.** Невозможно постоянно заниматься только тестированием. Руководители разработки постоянно требуют ресурсов и внимания тестирования, поэтому тест-менеджеры должны уметь вежливо и обоснованно отказывать.
- **Внешние коммуникации.** Менеджерам доводится работать с внешними исполнителями, нанятыми для временной работы в проекте или для тестирования продуктов извне. В зоне их ответственности организация мероприятий для обсуждения и обмена новыми инженерными практиками в области тестирования с широкой публикой. Живой пример тому — конференция GTAC.

- **Стратегические инициативы.** Менеджмент тестирования часто спрашивают, что еще такого можно сделать в Google, чего нельзя сделать нигде больше? Как мы можем расширить свою тестовую инфраструктуру и поделиться ею, чтобы улучшить интернет в целом, а не только наш продукт или компанию? Что будет, если мы объединим наши ресурсы и возьмемся за долгосрочные проекты? Поддержка таких инициатив идеями, финансами и защитой от натиска команд, желающих срочно заполучить тестировщика, — это работа, требующая огромной отдачи.
- **Аттестация и оценка персонала.** Чтобы оценивать своих сотрудников, менеджеры в Google опрашивают всех участников команд. Результаты сравнивают между командами, чтобы выровнять их. Всех сотрудников Google оценивают раз в квартал. Мы делаем упор на то, что человек сделал за последнее время и как он повлиял на качество продукта, что он сделал для команды и для пользователей.

Наша система аттестации не позволяет сотруднику выезжать на прошлых успехах, важно постоянное развитие. Механика оценки эффективности все еще формируется, поэтому мы не будем подробно ее описывать. В общих чертах: сотрудники Google составляют краткое описание того, над чем они работали и как хорошо, по их мнению, они справились. Их коллеги и начальство добавляют комментарии, а затем на собраниях независимых комиссий проводится оценка и согласование результатов между командами. Мы ожидаем, что наши сотрудники Google будут ставить себе цели выше, чем они считают возможным достичь. Поэтому если сотрудник выполнил все намеченное, значит планка была не так уж высока.

Оценивая людей таким способом, мы выявляем и поддерживаем ребят, которым пора сменить специализацию на более подходящую. Перемещения происходят постоянно и во всех направлениях. Чаще всего инженеры по тестированию переходят в разработчики в тестировании, а разработчики в тестировании — в разработчики. Ребята прокачиваются технически и специализируются на своих новых интересах. На втором месте — перемещения из роли разработчика в тестировании на роль инженера по тестированию и из роли инженера по тестированию на роль руководителя проекта. Это легко объяснить тем, что людям надоедает целыми днями программировать и хочется расширить поле деятельности.

Руководители помогают людям устанавливать ежеквартальные и годовые OKR¹. Они следят за тем, чтобы цели ставились высокие и амбициозные, поощряют людей мечтать высоко даже при краткосрочном планировании. Менеджеры следят,

¹ OKR (Objectives and Key Results, то есть «цели и ключевые результаты») — это список целей и оценка успеха в достижении этих целей. Google уделяет большое внимание количественно измеряемым метрикам успеха. То есть достижение 70-процентного успеха означает, что вы поставили достаточно высокие цели и усердно работали для их достижения, а 100-процентный успех говорит о том, что вы были недостаточно амбициозны.

чтобы цели отвечали способностям и интересам инженеров и не конфликтовали с требованиями проекта и бизнеса.

Управлять тестированием — значит находить компромиссы и прислушиваться к каждому человеку. Управлять в Google — значит направлять и опекать своих людей, а не командовать ими. Возвращаясь к аналогии с пиратами: хороший руководитель обеспечивает семь футов под килем и попутный ветер.

Тестирование в режиме сопровождения

Google известен своими ранними и частыми выпусками, а еще — стремлением понять, что проект провальный, как можно быстрее. Поэтому мы можем срочно перебросить ресурсы на проект с наибольшими рисками. Что это значит для тестировщика? Фичи, над которыми он работает, а иногда и целые проекты могут потерять приоритет или полностью свернуться. Как с технической, так и с эмоциональной стороны инженер должен быть готов к этому. Да, возможно, вам придется бросить все, когда вы только что расставили все по полочкам, и начать сначала. Это не самые приятные моменты, и изменения могут обернуться высокими рисками и затратами, если действовать неосторожно.

Пример режима сопровождения: Google Desktop

Джейсон Арбон

На середине очередного проекта мне предложили взяться за колossalную задачу тестирования Google Desktop с десятками миллионов пользователей, клиентскими и серверными компонентами и интеграцией с поиском Google. Я стал последним в длинном списке тест-лидов этого проекта с типичными проблемами качества и техническим долгом. Проект был внушительным, но, как и в большинстве проектов такого объема, наращивание функциональности постепенно замедлилось, а за несколько лет тестирования и использования сократились и риски.

Когда мы с двумя коллегами-тестировщиками влились в работу над проектом, у Google Desktop уже было около 2500 тест-кейсов в старой базе данных Test Case Manager, а несколько старательных подрядчиков в хайдарабадском

офисе прогоняли эти тест-кейсы для каждого выпуска. Циклы прохождения тестов часто занимали по неделе и больше. Раньше уже были попытки автоматизировать тестирование продукта через пользовательский интерфейс, но они не удались из-за сложности проекта и их затратности. Управлять веб-страницами и пользовательским интерфейсом окна рабочего стола через C++ было непросто, постоянно возникали проблемы с задержками.

Моими коллегами-тестировщиками были Теджас Шах и Майк Мид. Тогда в Google ресурсы клиентского тестирования были ограничены. Так как большинство продуктов уже работало в веб или быстро туда переходило, мы решили взять фреймворк тестирования на Python, который разработали для Google Talk Labs Edition. Он управлял продуктом через веб-модель DOM. В этом простом фреймворке были реализованы основные средства — например, класс тест-кейсов, производный от PyUnit. Многие тестировщики и разработчики знали Python, так что у нас была стратегия выхода на всякий случай. А если бы что-то сломалось, многие другие инженеры могли помочь. Кроме того, Python прекрасно подходит для итеративной разработки маленьких фрагментов кода без компиляции. В Google он устанавливается на рабочие станции инженеров по умолчанию, так что весь тестовый пакет мог быть развернут одной командной строкой.

Мы с ребятами решили применить весь диапазон возможностей Python API для управления продуктом. Мы использовали стуре для управления COM API на стороне клиента для поиска, мы моделировали ответы сервера для тестового внедрения локальных результатов в результаты google.com (нетривиальная задача!), работали с множеством библиотечных функций, чтобы имитировать пользователя и воссоздать его путь в приложении. Мы даже построили систему виртуальных машин для автоматизации тестов, которым нужны индексы Google Desktop. Иначе нам пришлось бы несколько часов ждать завершения индексирования на новой установке. Мы сформировали небольшой автоматизированный пакет смоук-тестирования¹, покрывающий самые важные функции продукта.

Потом мы принялись изучать старые тест-кейсы. Многие из них было сложно даже понять. Например, там были ссылки на код из прототипов или уже удаленных функций ранних версий или же они сильно зависели от контекста и состояния машин. К сожалению, большая часть документации жила только в головах разработчиков из Хайдарабада. Если бы нам вдруг понадобилось быстро проверить сборку с фиксами безопасности, у нас бы ничего не

¹ Смоук-тестирование — это минимальный набор тестов, который проводят, чтобы убедиться, что базовые функции программы работают корректно. Если система не проходит смоук-тестирование, то дальнейшие тесты не имеют смысла. Термин родился в ходе проверки радиоэлектронных устройств — если после пробного включения детали аппарата перегреваются, «идет дым», то все устройство нужно отправлять на доработку. — Примеч. перев.

вышло. Это обошлось бы слишком дорого. Поэтому мы решились на смелый шаг — проанализировали все 2500 тестов, выявили самые важные и актуальные из них, а потом удалили все, оставив примерно 150 тестов. Да, количество тест-кейсов удалось здорово сократить. Поработав вплотную с подрядчиками, мы довели до ума оставшиеся ручные тест-кейсы — последовательность действий стала настолько ясной и подробной, что ее мог выполнить любой, кто пользовался Google Desktop хотя бы несколько минут. Мы не хотели остаться единственными людьми, которые могли выполнить регрессионную серию.

К этому моменту у нас было покрытие автоматизированными тестами для сборок, которое уже начало вылавливать регрессионные баги за нас, и очень небольшой набор ручных тестов, которые мог выполнить кто угодно за день для релизной версии. Все это дало подрядчикам время заняться более приоритетными вещами, затраты сократились, а задержки выпуска заметно уменьшились при практически неизменном функциональном покрытии.

Примерно тогда же начался проект Chrome, и это направление считали будущим для сервисов Google на клиентских машинах. Мы уже были готовы пожинать плоды наших трудов по автоматизации с полнофункциональным тестовым API, занесли руку над построением генерируемых тестов для длительного выполнения, но тут пришло распоряжение быстро перебросить ресурсы на браузер Chrome.

С автоматическими регрессионными тестами, проверявшими каждую внутреннюю и публичную сборку, с очень простым ручным тестированием нам не стыдно было оставить Desktop в режиме сопровождения и сосредоточиться на менее устойчивом и рискованном проекте Chrome.

Но у нас оставался еще один капризный баг: в некоторых версиях Google Desktop у некоторых пользователей поглощалось свободное пространство на диске. Решение проблемы откладывалось, потому что мы не могли воспроизвести его. Мы попросили своего коммюнити-менеджера собрать больше информации от пользователей, но никто не смог четко определить проблему. Мы переживали, что со временем этот баг мог затронуть большее количество пользователей, а без полноценной команды на проекте он не будет решен вообще. Поэтому перед уходом мы бросили все силы на исследование этой проблемы. Дошло до того, что автора исходного кода индексирования выдернули на проект из другого места, полагая, что он знает, где нужно искать. И он знал. Разработчик заметил, что Desktop продолжает повторно индексировать задачи, если у пользователя установлен Outlook. Дело в том, что код индексирования при каждом сканировании принимал старую задачу за новую и медленно, но верно пожирал жесткий диск. А предпочитал этот гурман только диски тех пользователей, которые пользовались планировщиком задач Outlook. Так как максимальный размер индекса 2 Гбайта, он заполнялся очень медленно, и пользователи

замечали только то, что их новые документы не проиндексировались. Мы выпустили последнюю версию Desktop с исправлением этого бага, чтобы нам не пришлось возвращаться к этой проблеме через полгода на проекте с урезанной командой.

Мы также заложили бомбу с часовым механизмом под одну фичу, предупреждая пользователей о том, что скоро она перестанет существовать. Команда тестирования предложила отказаться от варианта с обращением к серверу для проверки глобального флага, указывающего на отключение фичи, в пользу более надежного и стабильного решения, работающего только на стороне клиента. Так нам не нужно было выпускать новую версию с выключенной функцией, а простота решения сделала работу функции более надежной.

Мы написали краткую документацию и практическое руководство по выполнению автоматизации и запуску ручных тестов. Теперь для небольшого релиза нужно было всего несколько часов одного тестировщика-контрактника. Назначили исполнителя и перенаправили свои усилия по тестированию на Chrome и облачные технологии. Все последующие инкрементальные запуски проходили и проходят до сих пор как по маслу, а продукт продолжает активно использоваться.

При переводе проекта в режим сопровождения качества важно снизить зависимость от участия людей в поддержании качества. У кода есть интересное свойство: если оставить его без внимания, он черствеет и ломается сам по себе. Это верно и для кода продукта, и для кода тестов. Поэтому при организации поддержки ставьте на первое место контроль качества, а не выявление новых проблем. Как и везде, если ресурсов много, набор тестов не всегда минимален, поэтому нужно сфокусироваться на главном и смело отсечь все лишнее.

При исключении ручных тестов мы руководствуемся следующими правилами:

- Мы ищем тесты, которые всегда проходят, или тесты с низким приоритетом. Это наши первые кандидаты на выбывание!
- Мы понимаем, что именно выбрасываем. Мы не жалеем времени и делаем несколько репрезентативных выборок тестовых данных из исключаемых областей. Мы стараемся поговорить с авторами тестов, чтобы лучше понять их цели и не выплеснуть ребенка вместе с водой.
- Освободившееся время мы используем для автоматизации, анализа высокоприоритетных тестов или исследовательского тестирования.
- Мы отсекаем автоматизированные тесты, которые давали ложные положительные срабатывания в прошлом или просто ненадежны, — они могут поднимать ложную тревогу и приводить к лишней трате сил инженеров.

Несколько советов, которые лучше бы учесть перед переходом в режим сопровождения.

- Не бросайте серьезные проблемы, исправьте их, прежде чем покинуть проект.
- Даже самый маленький автоматизированный пакет сквозных тестов может обеспечить долгосрочную уверенность при почти нулевых затратах. Сделайте такой пакет, если у вас его еще нет.
- Оставьте документ-инструкцию, чтобы любой сотрудник компании мог выполнить ваш пакет тестов. Это спасет вас от отвлекающих звонков с разнообразными вопросами в самый неподходящий момент.
- Убедитесь в том, что у вас есть схема эскалации при возникновении проблемы. Оставьте и себе место в этой схеме.
- Всегда будьте готовы оказать помощь проектам, над которыми вы когда-то работали. Это полезно для продукта, команды и для пользователей.

Переход в режим сопровождения — обязательный жизненный этап для многих проектов, особенно в Google. Мы, как инженеры по тестированию, обязаны сделать все для того, чтобы этот переход был безболезненным для пользователей и эффективным для команды разработки. Нужно двигаться дальше, не привязываясь к своему коду или прежним идеям, — научитесь отпускать их во взрослую жизнь.

Эксперимент с Quality Bots

Как изменится тестирование, если мы забудем о наших методах и инструментах, а возьмем на вооружение инфраструктуру поисковых движков? Почему бы и нет, ведь там есть бесплатное процессорное время, свободное пространство и дорогая система вычисления, работающая над алгоритмами! Давайте поговорим о ботах, а конкретнее — о Quality Bots.

Завершив множество проектов по тестированию в Google и поговорив со многими командами, мы осознали, что блестящие умы наших инженеров часто растратчиваются на ручное построение и выполнение регрессионных тестов. Поддерживать автоматизированные тестовые сценарии и вручную проводить регрессионное тестирование — дорогое удовольствие. К тому же медленное. Добавляет масла в огонь то, что мы проверяем ожидаемое поведение, — а как же неожиданное?

Регрессионные тесты обычно проходят без ошибок в более чем 95% случаев. Скорее всего, так происходит потому, что практика разработки в Google заточена на качество. Но, что важно, эта рутинная работа притупляет способности инженеров, которых мы нанимали вообще-то за любознательность и изобретательность. Мы

хотим освободить наших ребят для более сложного, исследовательского тестирования, для которого, собственно, мы их и брали в команду.

Google Search постоянно сканирует веб-пространство: запоминает, что видит, упорядочивает и ранжирует полученные данные в огромных индексах, руководствуясь статической и динамической релевантностью (качеством информации), а потом выдает информацию по запросу на странице результатов поиска. Если хорошенько подумать, базовая архитектура системы поиска может быть отличным примером автоматизированной системы оценки качества. Выглядит как идеальный движок для тестирования. Мы не стали два раза вставать и построили себе версию этой системы.

1. **Обход.** Боты работают в вебе¹ прямо сейчас. Тысячи виртуальных машин, вооруженные скриптами WebDriver, открывают в основных браузерах популярные URL-адреса. Перепрыгивая от одного URL-адреса к другому, словно обезьянки с лианы на лиану, они анализируют структуру веб-страниц, на которые приземляются. Они строят карту, которая показывает, какие элементы HTML отображаются, где и как.
2. **Индексирование.** Боты передают сырье данные серверам индексирования, где информация упорядочивается по типу браузера и времени обхода. Формируется статистика о различиях между обходами, например количество обойденных страниц.
3. **Ранжирование.** Когда инженер хочет посмотреть результаты для конкретной страницы по разным обходам или результаты всех страниц для одного браузера, система ранжирования вычисляет оценку качества. Проще говоря, система оценивает сходство страниц в процентах: 100% означает, что страницы идентичны. Соответственно, чем меньше процент сходства, тем больше различий.
4. **Результаты.** На информационной панели можно посмотреть сводку результатов (рис. 3.27). Подробные результаты строятся в виде простой таблицы оценок для каждой страницы с указанием сходства в процентах (рис. 3.28 и 3.29). Для каждого результата инженер может копнуть глубже и получить информацию о визуальных различиях. Они показаны с помощью наложения результатов разных проходов с указанием XPath-путей² элементов

1 Самые приоритетные обходы выполняются на виртуальных машинах Skytap.com. Это мощная сеть виртуальных машин. Она позволяет разработчику напрямую связаться с той машиной, на которой произошел сбой, и управлять отладкой, даже не выходя из браузера. Время и внимание намного ценнее вычислительных процессов. Skytap позволяет ботам работать полностью на сторонних виртуальных машинах и аккаунтах, открывая им доступ к непубличным промежуточным серверам.

2 Пути XPath похожи на пути к файлам, но используются в веб-страницах, а не в файловых системах. Они идентифицируют отношения «родитель/потомок» и другие сведения, однозначно определяющие элемент в DOM-дереве веб-страницы. См.: <http://ru.wikipedia.org/wiki/Xpath>

и их позиций (рис. 3.30). Инструмент показывает средние минимальные и максимальные исторические показатели этого URL-адреса и другую по-добную информацию.

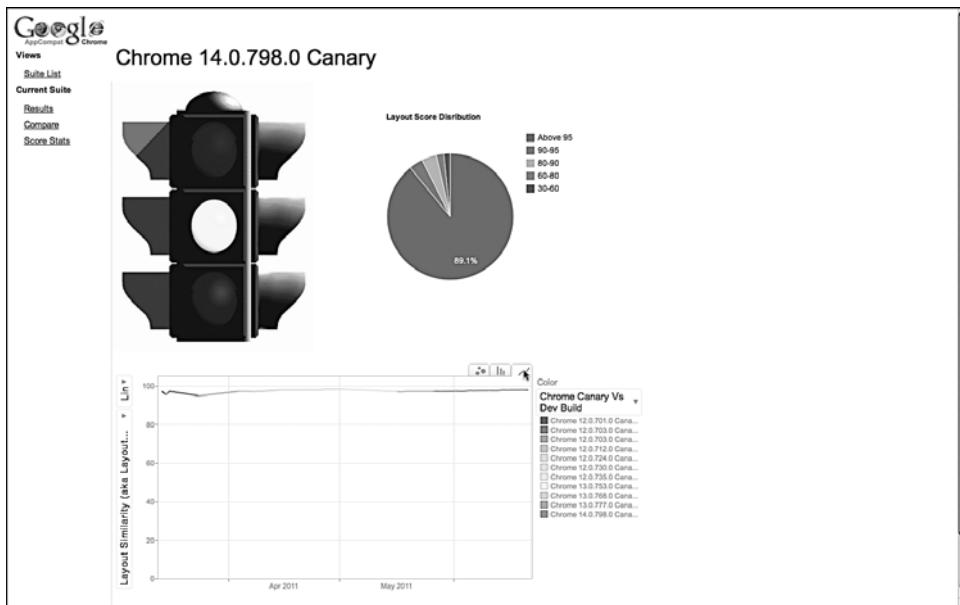


Рис. 3.27. Сводка информации для разных сборок Chrome

The screenshot shows a detailed results table for a suite run on June 21, 2011, at 04:21:55 PM. The table has columns for Test Browser, Ref Browser, Url, Score, Date, Comments, and Bugs. The table lists 19 rows of data, each corresponding to a different URL and its test results. The "Comments" and "Bugs" columns are mostly empty or show "None". The "Score" column shows values ranging from 91.6279 to 100.0000. The "Date" column shows the run date and time. The "Test Browser" column shows "Chrome 14.0.798.0 (wicked_fast)" and the "Ref Browser" column shows "Chrome 14.0.798.0".

Test Browser	Ref Browser	Url	Score	Date	Comments	Bugs
Chrome 14.0.798.0 (wicked_fast)	Chrome 14.0.798.0	http://www.tribalfusion.com	100.0000	Tue 21 Jun, 2011 11:19:40 PM UTC		
Chrome 14.0.798.0 (wicked_fast)	Chrome 14.0.798.0	https://www.wikia.com	100.0000	Tue 21 Jun, 2011 11:18:48 PM UTC		
Chrome 14.0.798.0 (wicked_fast)	Chrome 14.0.798.0	http://www.guardian.co.uk	99.2204	Tue 21 Jun, 2011 11:18:39 PM UTC		
Chrome 14.0.798.0 (wicked_fast)	Not Pre-rendered	Chrome 14.0.798.0	99.6812	Tue 21 Jun, 2011 11:18:33 PM UTC		
Chrome 14.0.798.0 (wicked_fast)	Chrome 14.0.798.0	http://www.koopa.com	91.6279	Tue 21 Jun, 2011 11:17:26 PM UTC		
Chrome 14.0.798.0 (wicked_fast)	Chrome 14.0.798.0	http://www.justin.tv	100.0000	Tue 21 Jun, 2011 11:17:23 PM UTC		
Chrome 14.0.798.0 (wicked_fast)	Chrome 14.0.798.0	http://www.reuters.com	100.0000	Tue 21 Jun, 2011 11:17:21 PM UTC		
Chrome 14.0.798.0 (wicked_fast)	Chrome 14.0.798.0	http://www.geocities.jp	99.9446	Tue 21 Jun, 2011 11:17:21 PM UTC		
Chrome 14.0.798.0 (wicked_fast)	Chrome 14.0.798.0	http://www.bild.de	100.0000	Tue 21 Jun, 2011 11:15:08 PM UTC		
Chrome 14.0.798.0 (wicked_fast)	Not Pre-rendered	Chrome 14.0.798.0	100.0000	Tue 21 Jun, 2011 11:15:07 PM UTC		
Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	100.0000	Tue 21 Jun, 2011 11:15:06 PM UTC		
Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	100.0000	Tue 21 Jun, 2011 11:13:53 PM UTC		

Рис. 3.28. Типичная таблица с подробной информацией от ботов

Google Chrome Suite: Tue 21 Jun, 2011 04:21:55 PM							
Views	<ul style="list-style-type: none"> Number of URLs: ~447 Reference browser: Chrome 14.0.798.0 Test browsers: Chrome 14.0.798.0 Data received: 895 Result computed: 424 						
Suite List							
Current Suite							
Results							
Compare							
Score Stats							
Latest Suite							
Pass							
Fail							
0 - 19	Next 20 >						
		Sort: Latest Score High > Low					
	Test Browser	Ref Browser	Url	Score	Date	Comments	Bugs
	Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	http://www.as.com	18.7728	Tue 21 Jun, 2011 06:09:21 PM UTC	Banner ad in the middle issue.
	Chrome 14.0.798.0 (wicked_fast)	Not Pre-rendered	Chrome 14.0.798.0	http://www.blogbus.com	18.8542	Tue 21 Jun, 2011 05:06:07 PM UTC	Data screw up - needs investigation.
	Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	http://www.nifty.com	21.0968	Tue 21 Jun, 2011 07:23:10 PM UTC	None
	Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	http://www.soufun.com	42.2505	Tue 21 Jun, 2011 07:10:14 PM UTC	None
	Chrome 14.0.798.0 (wicked_fast)	Not Pre-rendered	Chrome 14.0.798.0	http://www.reddit.com	45.3753	Tue 21 Jun, 2011 05:23:17 PM UTC	Too much dynamic content issue.
	Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	http://www.google.pt	47.9971	Tue 21 Jun, 2011 11:11:08 PM UTC	Sync and change of page content issue.
	Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	http://www.google.be	48.8929	Tue 21 Jun, 2011 10:53:39 PM UTC	Sync issue , page content changed.
	Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	http://www.clarin.com	50.6132	Tue 21 Jun, 2011 06:03:47 PM UTC	Banner ad in the middle issue.
	Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	http://www.virgilio.it	52.6742	Tue 21 Jun, 2011 04:44:17 PM UTC	Top banner ad. Pixel shift.
	Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	http://www.duowan.com	56.1918	Tue 21 Jun, 2011 05:13:13 PM UTC	None
	Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	http://www.kub.com	62.8598	Tue 21 Jun, 2011 07:23:02 PM UTC	None
	Chrome 14.0.798.0 (wicked_fast)	Pre-rendered	Chrome 14.0.798.0	http://www.amazon.co.jp	63.4265	Tue 21 Jun, 2011 06:58:20 PM UTC	None

Рис. 3.29. Таблица информации от ботов, отсортированная для выявления наибольших различий



Рис. 3.30. Анализ визуальных различий для идентичных страниц

Первый же официальный запуск проекта нашел различие между двумя канаречными сборками Chrome. Боты провели проверку автоматически. Тестировщик оценил результаты и заметил, что этот URL-адрес потерял несколько процентов

сходства. Тестировщик быстро сообщил о проблеме, ссылаясь на подробную картинку (рис. 3.31) с выделенной частью страницы с различиями. Раз боты могли протестировать все версии Chrome¹, инженер мог быстро справляться с новыми регрессионными багами. Каждая сборка содержала всего несколько изменений, и заливку с проблемным кодом оперативно изолировали. Оказалось, что коммит² в репозиторий WebKit (ошибка 56859: reduce float iteration in logicalLeft/RightOffsetForLine) вызвал регрессионный баг³, из-за которого средний элемент div на этой странице стал отображаться ниже границы страницы. Тестировщик завел баг 77261: Макет страницы ezinearticles.com неправильно отображается в Chrome 12.0.712.0.

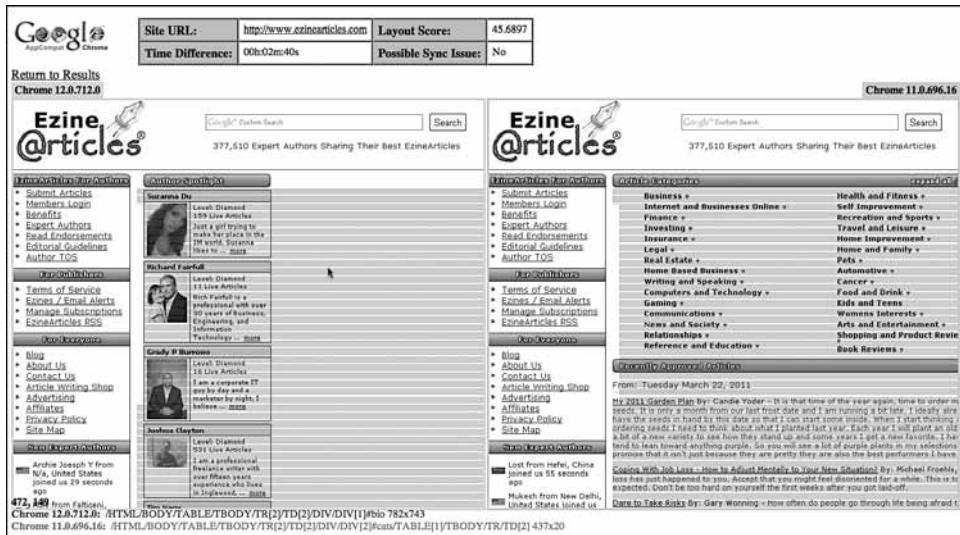


Рис. 3.31. Первый баг, обнаруженный при первом запуске ботов

Как мы прогнозировали (и надеялись), данные от ботов оказались очень похожи на данные, получаемые от их одушевленных аналогов, и во многом даже лучше. Большинство веб-страниц оказывались идентичными в разных версиях браузеров, и даже если находилось различие, инженер быстро просматривал его и понимал, есть ли что-то серьезное.

Машины теперь могли автоматически подтвердить отсутствие регрессионных багов. Это маленький шаг для машины, но огромный для всего мира тестировщиков — им больше не нужно пробираться через тернии не самых интересных страниц. Тесты теперь можно прогонять за минуты, а не за несколько дней, как

- Сборки Chrome появляются несколько раз в день.
- О том, что вызвало эту регрессию, можно узнать по URL-адресу <http://trac.webkit.org/changeset/81691>
- URL-адрес проблемы WebKit Bugzilla: https://bugs.webkit.org/show_bug.cgi?id=56859. Адрес ошибки в Chromium: <http://code.google.com/p/chromium/issues/detail?id=77261>

раньше. Их можно проводить ежедневно, а не еженедельно. У тестировщиков на конец освободились руки и время и стало возможным заняться багами посложнее.

Если оставить версию браузера неизменной, а менять при этом только данные одного сайта, мы получим средство для тестирования сайтов, а не только браузера. Такую же штуку можно провернуть с анализом одного URL-адреса по всем браузерам и всем сериям тестов. То есть у веб-разработчика появилась возможность просмотреть все изменения, происходящие с его сайтом: он создает новую сборку, дает ботам ее обойти и получает таблицу результатов, где показаны все изменения. Быстро, безо всякого ручного тестирования, веб-разработчик определяет, какие изменения из обнаруженных не заслуживают внимания, а какие похожи на регрессионный баг и достойны занесения в багтрекинговую систему, причем сразу с информацией о браузерах, версии приложения и конкретных элементах HTML, где он водится.

А как насчет веб-сайтов, управляемых данными? Возьмем, например, сайты YouTube и CNN — их контент огромен и изменяется со временем. Не запутаются ли боты? Они справятся, если будут предупреждены о нормальных колебаниях данных этого сайта. Например, если в нескольких последовательных сериях изменился только текст статьи и картинки, то боты посчитают изменения уместными для данного сайта. Если показатели выйдут за рамки (допустим, при нарушении IFRAME или при переходе сайта на другой макет), боты могут подать сигнал тревоги и сообщить об этом веб-разработчику, чтобы он определил, нормально ли новое состояние или пора заводить соответствующий баг. Пример небольшого шума можно увидеть на рис. 3.32: на сайте CNET есть реклама, которая во время проверки

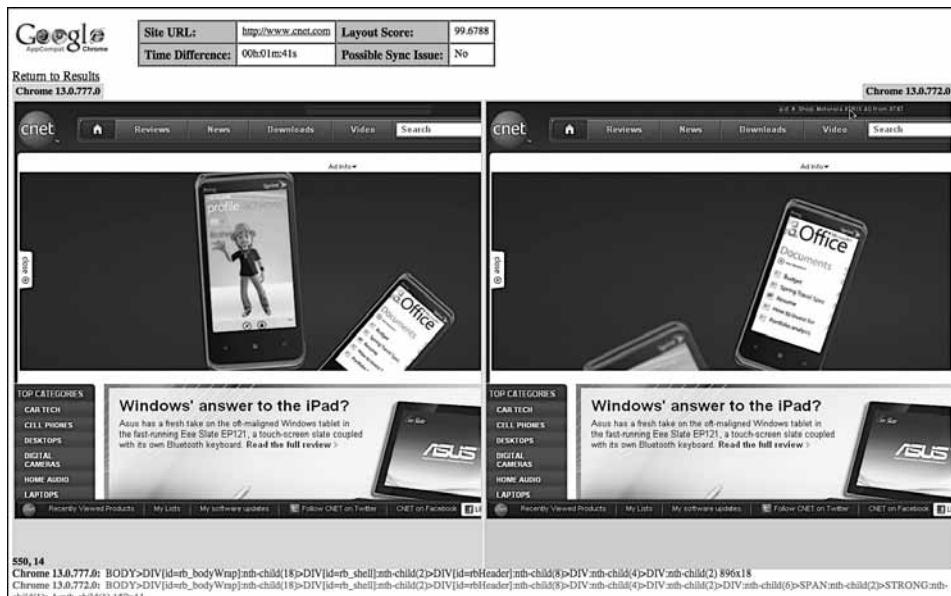


Рис. 3.32. Анализ визуальных различий для страниц с шумовыми различиями

появилась справа, а не слева. Такой шум считается небольшим и будет либо проигнорирован ботом, либо помечен как несущественный человеком, который моментально заметит, что это всего лишь реклама.

А что происходит дальше со всеми этими сигналами? Должен ли тестировщик или разработчик просматривать их все? На самом деле нет, мы уже ведем эксперименты по прямой передаче информации о различиях краудсорс-тестировщикам¹, чтобы они быстро ее проверяли. Мы хотим оградить наши основные команды разработки и тестирования от лишнего шума. Мы просим внешних помощников посмотреть две версии веб-страницы и обнаруженные различия. Они отмечают, баг это или несущественное отклонение.

Как мы получаем данные от сообщества? Гениальное — просто: мы построили инфраструктуру, которая транслирует необработанные данные ботов на обычную страницу голосования для тестировщиков. Разумеется, мы сравнивали работу краудсорсеров со стандартными методами ручного рецензирования. Схема была следующая: боты пометили только шесть URL-адресов как требующие дополнительной проверки. Помеченные URL-адреса получили тестировщики из сообщества. Имея в арсенале данные ботов и инструменты визуализации различий, краудсорсеры определяли, ошибка ли это, в среднем за 18 секунд. А проверка всех 150 URL-адресов на регрессию ручными методами заняла около трех дней. Тестировщики из сообщества успешно определили все шесть различий как несущественные. Результаты работы краудсорсеров и ручной затратной формы проверки совпали! А зачем платить больше?

Звучит здорово! Правда, этот метод подходит только для статических версий веб-страниц. А как насчет интерактивных элементов — раскрывающихся меню, тестовых полей и кнопок? Мы ведем работу по решению этой проблемы, можно сказать, мы открыли киностудию: боты автоматически взаимодействуют с интересующими нас частями веб-страницы и снимают на каждом шаге кадр DOM. Затем «фильмы» каждой серии сравниваются покадрово с помощью той же технологии анализа различий.

В Google некоторые команды уже заменили большую часть своей ручной работы по регрессионному тестированию ботами. У них появилось время для более интересной работы, например исследовательского тестирования, которой они не могли заниматься раньше. Команда поставила себе цель сделать сервис общедоступным, выложить исходный код для всех и добавить возможности собственного хостинга, чтобы команды могли тестировать внутри своей сети, если кто-то не хочет открывать свои URL-адреса наружу. Мы не торопимся с массовым внедрением новой технологии — нужно убедиться в ее стопроцентной надежности.

Базовый код проекта Bots работает в инфраструктурах Skytap и Amazon EC2. Код сервиса распространяется по модели открытого кода (подробнее в блоге те-

¹ Наши друзья из <http://www.utest.com> помогли в организации этих экспериментов. Тестировщики из этого сообщества чрезвычайно наблюдательны и отзывчивы. Иногда они находили больше ошибок, чем внутренние многократные запуски регрессионных тестов.

стирования Google и приложении B). Теджас Шах был техническим руководителем Bots с первых дней существования проекта; позднее к нему присоединились Эриэл Томас, Джо Михаил и Ричард Бустаманте. Присоединяйтесь и вы к этим ребятам, чтобы двигать эксперимент дальше!

Как оценить качество всего интернета

Чтобы измерить, насколько хорошо поисковая система справляется с запросами, для теста мы берем случайную репрезентативную выборку поисковых запросов. По результатам можно судить, как система будет работать со всеми запросами, — мы просто экстраполируем данные. А если мы используем Bots на репрезентативной выборке URL-адресов, мы можем судить о качестве интернета в целом.

Сингулярность¹: легенда о происхождении ботов

Джейсон Арбон

Давным-давно, в далеком-далеком офисе Google родилась... первая версия Chrome. Уже по первым поступившим данным было понятно, что Chrome отображает веб-страницы иначе, чем Firefox. В начале мы оценивали эти различия, только отслеживая объем поступающих сообщений о багах и подсчитывая количество жалоб на проблемы совместимости от пользователей, которые удаляли браузер после пробного использования.

Мне было интересно, есть ли более многоразовый, автоматизированный и объективный метод оценки того, насколько хорошо мы работаем в этой области. Были ребята до меня, которые пытались организовать автоматическое сравнение снимков веб-страниц между браузерами, а кто-то даже пытался использовать продвинутые методы распознавания изображений и границ. Правда, эти методы часто не работали, потому что между страницами всегда много различий, вспомните хотя бы о разных картинках в рекламе, меняющемся контенте и т. д. В базовых тестах макетов WebKit вычислялся хеш-код всего макета страницы (см. рис. 3.33). Поэтому когда обнаруживалась проблема, инженеры не имели понятия о том, что именно

1 Термин «сингулярность» часто используется для описания момента, в который компьютеры превзойдут человеческий интеллект. Это будет интересное время, и мы уже сегодня видим его приближение (http://en.wikipedia.org/wiki/Technological_singularity).

в приложении не работает, у них был только снимок ошибки. Многочисленные ложноположительные¹ срабатывания только прибавляли работы инженерам, вместо того чтобы уменьшать ее.

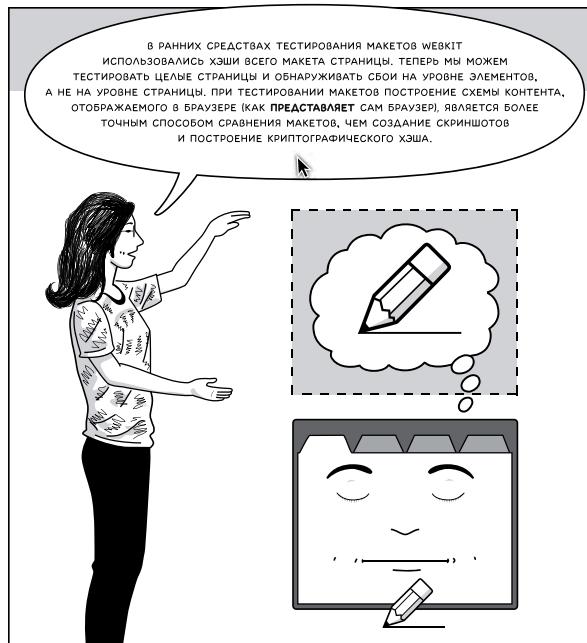


Рис. 3.33. В ранних средствах тестирования макетов WebKit использовались хэши всего макета страницы. Теперь мы можем тестировать целые страницы и обнаруживать сбои на уровне элементов, а не на уровне страницы

Мысли постоянно возвращали меня к ранней простой реализации Chromebot, которая обходила миллионы URL-адресов в запущенных копиях браузера Chrome на тысячах виртуальных машин. Она искала всевозможные сбои, используя для этого свободное процессорное время в центрах обработки данных. Это был ценный инструмент, который находил баги на ранней стадии, а функциональное тестирование взаимодействия с браузером добавлялось позже. К сожалению, технология утратила свою новизну и использовалась только для выявления редких сбоев. А что, если построить более серьезную версию этого инструмента, которая будет нырять во всю страницу целиком, а не только ходить по берегу? И назвать ее, например, Bots.

¹ Ложноположительными (false positives) называются сбои тестирования, вызванные не ошибками самого продукта, а ошибками тестового программного обеспечения. Обычно такие сбои обходятся дорого, раздражают инженеров и быстро снижают производительность их труда из-за безрезультатных исследований.

Я подумал об использовании другого подхода: работы в DOM¹. Около недели ушло на подготовку эксперимента, в котором загружалось много веб-страниц одна за другой, а потом в них внедрялся код JavaScript, который извлекал карту внутренней структуры веб-страницы.

Многие умные люди скептически отнеслись к этому решению. Они считали, что моя идея была обречена на неудачу, потому что:

- реклама постоянно изменяется;
- контент на таких сайтах, как CNN.com, постоянно меняется;
- специфичный для конкретного браузера код будет по-разному отображаться в разных браузерах;
- баги в самих браузерах будут приводить к возникновению различий;
- работа потребует огромных объемов данных.

Такая реакция сделала мою задачу только интереснее, и неудачи я не боялся. В прошлом я уже работал с другой поисковой системой, поэтому у меня была уверенность, что я смогу отделить сигнал от шума. К тому же в таком проекте у меня не было конкуренции. И я поднажал. В Google данные могут сказать много. И я хотел, чтобы они заговорили.

Чтобы запустить эксперимент, мне нужны были контрольные данные, с которыми я мог бы сравнивать полученные. Лучшим источником информации были тестировщики проекта Chrome, ведь они каждый день вручную открывали около 500 популярных сайтов в Chrome, пытаясь найти различия с Firefox. Я поговорил с подрядчиками, которые прогоняли вручную все эти сайты и сравнивали результаты с Firefox. Они рассказали, что сначала проблемы находились почти в половине популярных сайтов, но ситуация постепенно улучшалась, и сейчас расхождения встречаются редко — менее чем в 5% сайтов.

Я взял WebDriver (Selenium следующего поколения) и провел эксперимент. WebDriver лучше поддерживал Chrome, и его API куда более понятный. В первый прогон я собрал данные в разных версиях Chrome, от ранних до текущей, чтобы увидеть, найдет ли автоматизация такой же тренд. Тесты просто загружали те же веб-сайты, проверяли каждый пиксель, определяли, какой элемент HTML (не RGB-значение!) был видим в этой точке², а потом

1 DOM (Document Object Model) — внутреннее представление всего кода HTML, образующего веб-страницу. Модель DOM содержит все маленькие объекты, представляющие кнопки, текстовые поля, изображения и т. д.

2 getElementFromPoint(x,y) возвращал хэш элементов для секции веб-страницы размером 800 × 1000. С задачей можно было справиться более эффективно, но это решение было простым и хорошо иллюстрировало проблему.

отправляли данные на сервер. Выполнение на моем компьютере занимало около 12 часов, поэтому я запустил программу на ночь.

Полученные данные выглядели хорошо, поэтому я заменил Firefox на Chrome и снова запустил те же тесты на ночь. Конечно, мог появиться шум от изменения контента сайтов, но моя задача была только узнать, как выглядят данные, а потом выполнить обе серии параллельно. Когда я пришел утром в офис, я обнаружил, что мой компьютер выдернут из розетки. Мои соседи странно посматривали в мою сторону и сказали, что мне нужно поговорить со специалистами по безопасности. Я мог только догадываться, что они себе надумали. Оказалось, что во время обхода мой компьютер подхватил вирус с неизвестной сигнатурой, который и разбушевался ночью. Меня спросили, хочу ли я снять данные со своего компьютера, прежде чем диск будет уничтожен. К счастью, все мои данные хранились в облаке, и я отпустил свой компьютер с миром. После этого я стал запускать такие тесты только с внешних виртуальных машин.

За двое суток машины независимо выдали данные, которые были ужасно похожи на те, что мы получили примерно за год ручной тестовой работы (см. рис. 3.34). Подозрительно похожи.

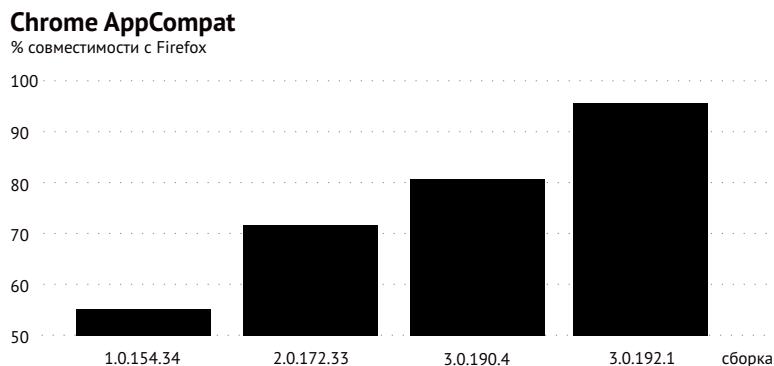


Рис. 3.34. Первые данные, демонстрирующие сходство между метриками количества, вычисленными людьми и ботами

Все это выглядело многообещающе. Результаты нескольких дней программирования и двух ночей выполнения на одном компьютере, кажется, сравнялись с результатами года работы группы тестировщиков. Я поделился своими данными с моим директором, имя которого называть не буду. Он посчитал, что это очень здорово, но предложил сосредоточиться на других, более зрелых проектах. Я поступил по-гугловски: сказал, что поставлю эксперимент на паузу, но делать этого не стал. Тем летом у нас была пара отличных интернов, которых мы подключили к оформлению этих запусков и поиску путей для более наглядного представления различий, — мы формировали

продукт. Они экспериментировали, замеряя разницу времени выполнения. Эрик Ву и Елена Янг продемонстрировали свою работу в конце лета и заставили всех поверить, что у нашего метода большое будущее.

Теджас Шах тоже оказался под впечатлением. Когда практиканты ушли от нас, Теджас создал инженерную команду, которая должна была превратить этот эксперимент в реальность.

Bots: детство, отрочество и масштабирование на весь интернет

Теджас Шах

Я — технический руководитель проекта Bots, я хочу масштабировать технологии Bots на весь интернет и открыть их миру. Проект Bots вырос из ранних экспериментов в полноценную технологию, которую используют многие команды в Google.

В конце 2010 года я работал над средой автоматизации для Chrome, известной как SiteCompact. Она использовала тесты JavaScript, чтобы автоматически находить функциональные баги в Chrome при просмотре популярных сайтов. Она проверяла как поиск на google.com, так и отображение статей на CNN почти для всех сборок Chrome. Система работала на «отлично», выявляя регрессионные баги и дополняла автоматизированные функциональные проверки поведения сайтов.

В то же время интерны Джейсона работали над демоверсией крутейшего проекта Bots. Я присматривал за ходом дела, но когда они продемонстрировали результаты, мои представления о том, как следует проверять сайты, изменились навсегда. Увидев демоверсию ботов, созданную Еленой, с первыми данными, я был покорен. Я понял, что передо мной возможность фундаментального изменения подхода к веб-тестированию. Мои скриптовые тесты, конечно, были важны, но они масштабировались только линейно, и их нужно было сопровождать. А проект Bots содержал в себе что-то более универсальное. Я сразу влюбился в эту технологию. Практиканты ушли, и все знали, что их код был только демонстрацией. Чтобы сделать его частью базовой инфраструктуры и решением, пригодным для веба, нужно было еще много работать.

Первые несколько месяцев я работал над Bots один. Я хотел избежать лишних вопросов и скептицизма. Но я верил, что есть тот, кто справится с задачей.

Какое-то время я работал в одиночку, избегая вопросов и скептических взглядов. Это продолжалось около квартала. Я проделал большую работу, решая задачи масштабирования, производительности, методов оценки и удобства использования страниц с различиями — пока все фрагменты не начнут работать как единое целое, пользы от системы не будет. Трудно решать такую задачу в одиночку, тем более зная, что работа над таким неоднозначным проектом — это риск для твоей карьеры. Если ничего не выйдет — тебе нечего будет показать. Google поощряет эксперименты, но хочет видеть результаты. Мое место в структуре Google ограждало меня от скептических вопросов во время аттестации, пока я работал над этим долгосрочным проектом.

Потом мы представили первую демоверсию руководителю разработки Chrome. Идея настолько захватила его, что он включил результаты Bots в повседневную работу по тестированию Chrome. Это признание сыграло важную для меня роль и придало мне уверенности, чтобы продолжать работу. А еще я понял, что если Chrome может использовать нашу систему, чтобы обнаруживать сложные проблемы, значит то же самое может делать любое веб-приложение.

Сразу же после этого мы провели презентации во многих командах Google. Каждый, кому мы показывали Bots, хотел использовать эту систему. Мы убеждались в реальности нашей мечты об использовании этой технологии во всех веб-приложениях. Поработав над ней еще несколько месяцев, я смог построить графики и трендов и результатов для канареекной сборки Chrome. Теперь Bots не только работала как система раннего оповещения, но и обнаруживала реальные баги на ранней стадии цикла. Система представляла куда более точные данные о сбоях, так что разработчики могли принимать решения на основе точных фактов. Мой любимый баг нашла первая боевая версия Bots, сравнив две сборки в один день. Bots обнаружила баг через несколько часов после того, как разработчик из Apple изменил атрибут WebKit. Фича была покрыта юнит-тестами, но только технология Bots смогла поймать этот баг, потому что тестировала реально существующие веб-страницы.

После презентации мою команду часто спрашивали: «Смогу ли я отказаться от ручного тестирования?» Наш ответ — твердое «нет». Тестировщики теперь могут выполнять работу, для которой их нанимали: исследовательское тестирование, анализ рисков и интересов пользователя.

Успех в Chrome привлек ресурсы в наш проект. Теперь у нас была пара инженеров, которые активно работали над Bots и помогали нам поднять продукт на следующий уровень. Тогда же нас попросили помочь команде поиска, которая находилась в процессе выпуска новой классной фичи Instant

Pages. Мы потратили на Instant Pages еще несколько недель, — нужно было научить систему запускать Chrome в разных режимах. Мы написали специальную серию тестов Bots, и теперь разработчики могли спокойно выпускать свой продукт, ведь они знали, что те же тесты пройдут автоматически для любых изменений, которые они внесут в будущем.

Мой совет инженерам по тестированию: если вы во что-то верите — делайте это! Мой совет менеджерам: не перекрывайте инженерам кислород, разрешите им экспериментировать, и они сотворят настоящие чудеса для вашего бизнеса и пользователей.

Эксперимент BITE

Мы создали BITE (Browser Integrated Test Environment), тестовую среду, интегрированную в браузер, для того чтобы вынести как можно больше тестовых действий, инструментов и данных в браузер и облако и показывать их в контексте. Мы хотели уменьшить время, которое тестировщики тратят не на тестирование, отвлекаясь от него. Мы хотели сделать тестирование еще более эффективным.

Что общего у пилота истребителя и тестировщика? Они оба тратят много времени на переключение контекста и обработку большого количества данных. В браузере у тестировщика часто открыто сразу несколько вкладок: одна с багтрекинговой системой, другая с электронной почтой проектной рассылки, третья с системой управления тест-кейсами, четвертая с планом тестирования. Наш пилот-тестировщик постоянно лавирует между этими вкладками. Может показаться, что мы чересчур зацикливаемся на скорости и эффективности, но здесь действительно есть проблема: легко потерять ценный контекст. Плюс ко всему:

- тестировщик тратит время на регистрацию дубликатов багов, потому что не знает правильных ключевых слов, чтобы найти уже существующие;
- тестировщик не заводит баги для проблем, которые кажутся очевидными, потому что не хочет рыскать по багтрекинговой системе в поисках правильного ключевого слова, чтобы убедиться в том, что такой баг уже занесен;
- не каждый тестировщик знает, где взять всю отладочную информацию, которая поможет разработчикам отсортировать и отладить баги;
- нужно время, чтобы вручную ввести, где был обнаружен баг, как его воспроизвести и другие важные для отладки данные. Эта рутинная работа часто выматывает и притупляет внимание инженера, как раз в тот момент, когда он должен быть особенно сконцентрирован на поиске багов.

BITE старается решить многие из перечисленных проблем и развязать тестировщику руки, дать ему сосредоточиться на самом тестировании, а не на механической работе.

Переместимся в кабину пилота истребителя. Проблема информационной перегруженности пилота решается с помощью индикаторов на лобовом стекле. Они упорядочивают информацию и подают ее в нужном контексте, как раз в поле зрения пилота. Чем сложнее становился самолет, тем больше появлялось данных и тем быстрее приходилось реагировать пилоту. Развитие разработки продуктов в Google происходит по той же схеме — чем дальше, тем больше выпусков, больше данных и тем быстрее нужно принимать решения. Мы позаимствовали подход у авиации, когда разрабатывали BITE для регрессионного и ручного тестирования.

Мы реализовали BITE как расширение браузера, чтобы можно было проследить за действиями тестировщика (см. рис. 3.35) и исследовать внутреннюю структуру веб-приложения. К тому же расширение позволяло показывать одинаковый для всех пользовательский интерфейс на панели инструментов браузера и быстро просматривать данные поверх веб-страницы: выглядит, как индикаторы на лобовом стекле пилота-тестировщика.



Рис. 3.35. Всплывающее окно расширения BITE

Давайте посмотрим, как эти экспериментальные возможности будут работать с реальными веб-приложениями Google.

Регистрируем баги с BITE

Помните про правило одного клика, которое работает в Google Feedback? Тестировщик, обнаружив баг в веб-приложении, может одним точным ударом сообщить о баге, выделив часть страницы, где возникла проблема, и дописав от себя сообщение. BITE, по сути, позволяет сделать то же самое, но в описание бага автоматически включается самая полезная и самая занудная для ручного ввода информация: URL-адрес, проблемный элемент или фрагмент текста на странице и снимок экрана. Для некоторых веб-приложений, в которых BITE встроен глубже, автоматически извлекаются отладочные URL-адреса и добавляется информация об отладке самой страницы.

Допустим, тестировщик ввел поисковый запрос «офисы Google» на maps.google.com и получил нерелевантный результат: Белый дом. Тогда тестировщик жмет в меню BITE кнопку «Сообщить о баге» и выделяет курсором часть страницы, где, по его мнению, находится баг: четвертый результат поиска в нашем случае (рис. 3.36). Он может выделить любые элементы управления, изображения, сектора карты, отдельные слова, ссылки или значки.

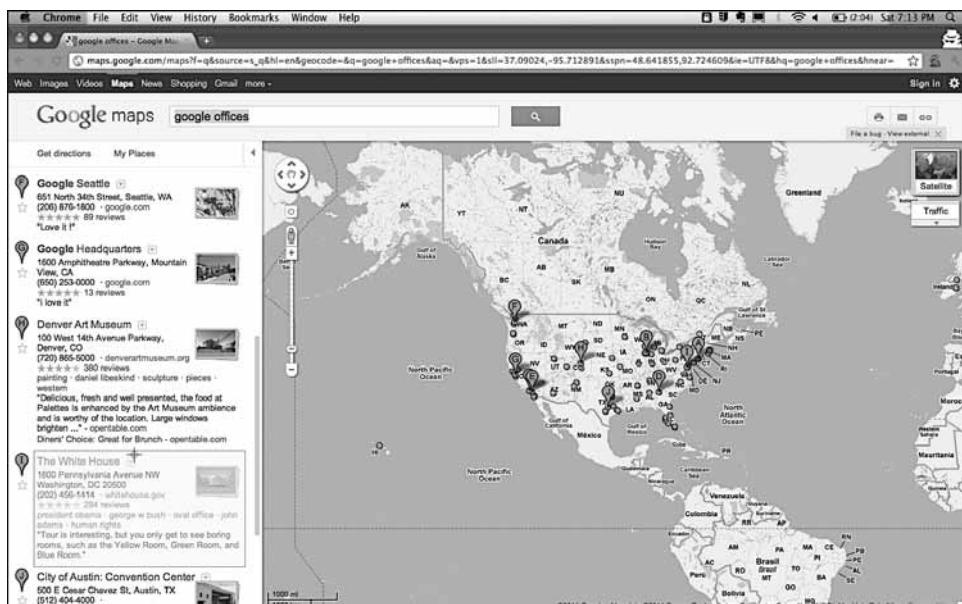


Рис. 3.36. BITE выделяет нерелевантный результат поиска — Белый дом, выделен в колонке слева

Если кликнуть по выделенному фрагменту страницы, откроется форма регистрации бага (рис. 3.37), и не нужно метаться с одной вкладки на другую. Тестировщик быстро вводит название бага и нажимает кнопку «Здесь баг!», чтобы быстро

добавить ошибку. Инженеры обычно не добавляют много данных, поэтому BITE добавляет их автоматически, и это здорово упрощает сортировку и отладку багов. Ну или от тестировщика нужно совсем немного усилий, но все равно он не сильно отвлекается от самого тестирования.

1. Снимок экрана создается автоматически и прикладывается к отчету о баге.
2. В отчет вкладывается код HTML выделенного элемента.
3. Все действия, совершенные с переходом на maps.google.com, записываются в фоновом режиме и преобразуются в код JavaScript. Если разработчик захочет понаблюдать, как воспроизведется этот баг в его браузере, ему достаточно открыть ссылку на код, которая будет автоматически приложена к отчету (обратите внимание на рис. 3.38).
4. Отладочные URL-адреса конкретной карты тоже будут автоматически прикреплены к отчету. Часто в обычных адресах недостаточно информации для полного воспроизведения.
5. Все данные о браузере и ОС тоже прикладываются.

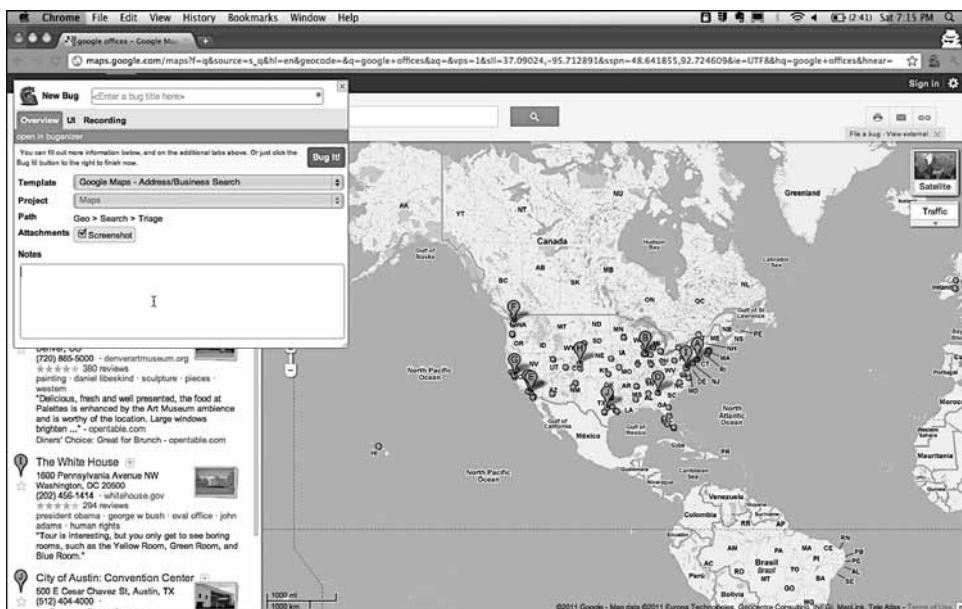


Рис. 3.37. BITE: встроенная форма сообщения о баге

Информация о баге заносится в багтрекинговую систему с полной информацией для приоритизации, и скорости регистрации багов позавидует любой пилот истребителя.

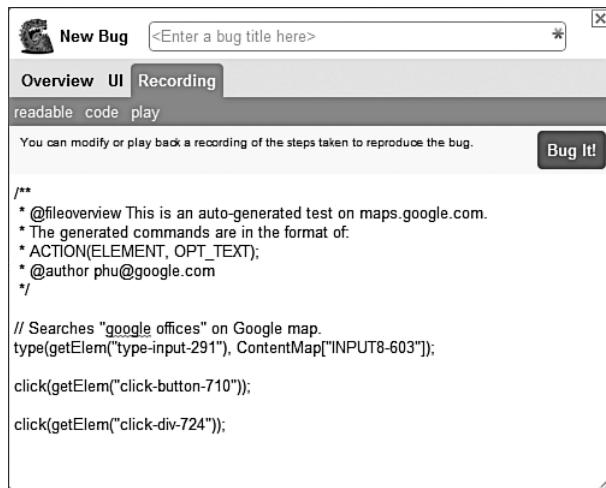


Рис. 3.38. BITE: код JavaScript, записанный в ходе тестирования

Влияние BITE на проект Maps

Сейчас BITE используют только внутри Google для регистрации багов Google Maps. Большая часть информации о состоянии приложения не сохраняется в URL-адресе, а серверные данные постоянно меняются, поэтому регистрация багов Google Maps — дело очень затейливое. Проще говоря, пользователи просматривают карты, меняют масштаб, но текущее состояние нигде не сохраняется. Когда появился BITE, менеджер продукта Google Maps был просто счастлив наконец снять с команды GEO этот груз. Он заверил нас, что теперь баги, поступающие от обычных сотрудников Google через BITE, ничем не уступают отладочной информации, получаемой от самых опытных тестировщиков, которые в Google Maps съели не одну собаку. Теперь приоритизация проходит быстрее, и разработчики могут воспроизводить и отлаживать намного больше багов, чем раньше. Без BITE они бы так и остались невоспроизводимыми.

Просмотр багов в BITE

Когда инженер в кабине пилота-тестировщика исследует приложение или выполняет регрессионные тесты, информация о багах страницы, на которой он сейчас находится, показывается прямо над тестируемым приложением. Это помогает быстро сориентироваться и понять, заведен ли уже этот баг и какие еще баги есть в этой части приложения.

BITE выводит информацию о багах как из внутренней базы, так и из системы отслеживания ошибок chromium.org, в которой внешние разработчики, тестировщики и пользователи могут заводить баги Chrome.

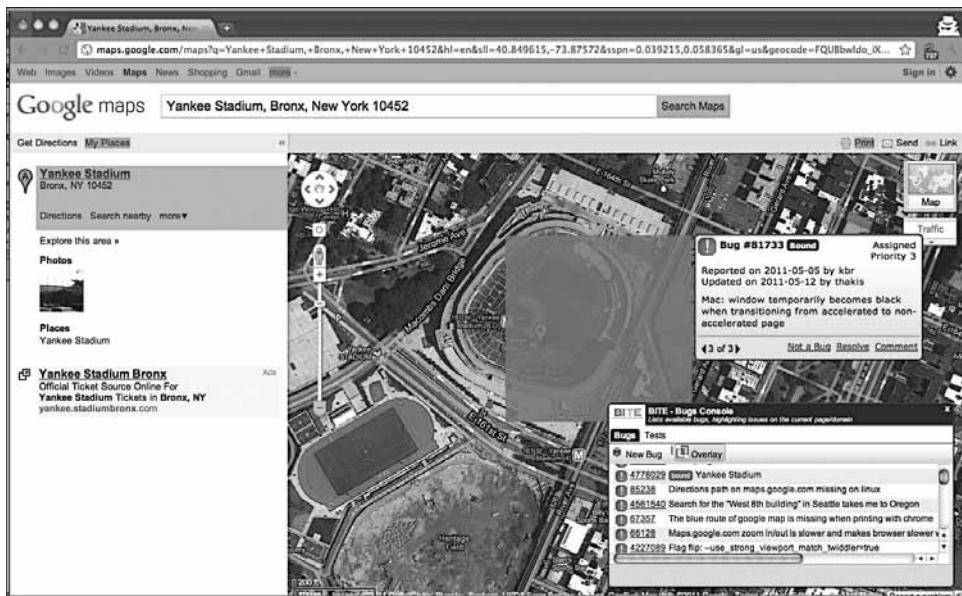


Рис. 3.39. BITE: панель с перечнем багов, относящихся к maps.google.com

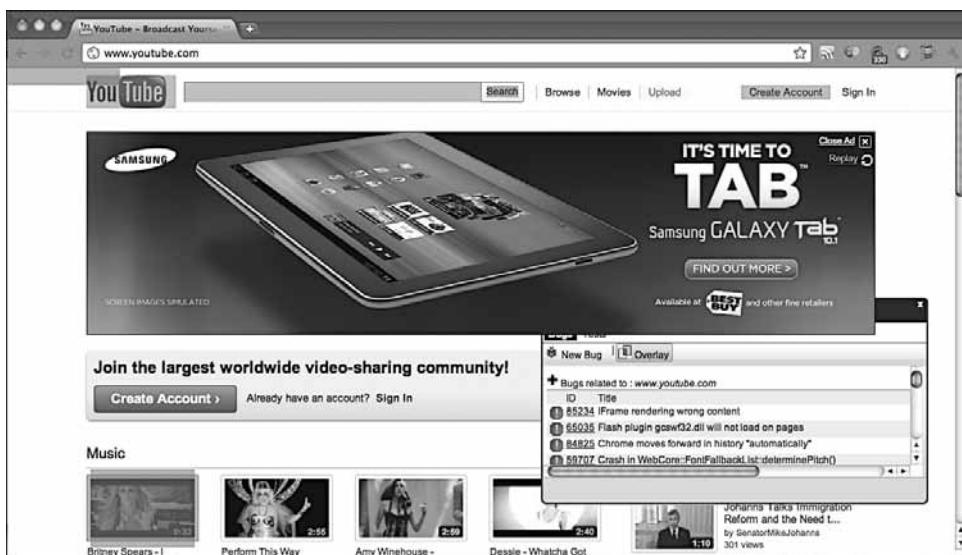


Рис. 3.40. BITE: панель багов на домашней странице YouTube

Число рядом со значком BITE в браузере указывает, сколько багов связано с текущей веб-страницей. Это очень просто делается для багов, которые завели через BITE. У нас есть все данные про них, вплоть до части страницы, где он проявился. А с багами, заведенным традиционным способом, например непосредственно в Issue Tracker или в нашей внутренней системе Buganizer, мы поступаем по-другому. У нас есть бот, который ищет URL-адреса в их описаниях и сравнивает степень их соответствия с URL-адресом текущей страницы. Сначала показываются точные совпадения, потом совпадения путей, а потом совпадения с доменом текущего URL-адреса. Схема простая, но очень надежная.

Рисунок 3.39 показывает, как выглядит страница карты с наложенной панелью багов BITE. Один клик на номер бага откроет полную страницу отчета в Buganizer или Issue Tracker. На рис. 3.40 показана панель багов на странице YouTube.

Запись и воспроизведение сценариев в BITE

Тестировщики и разработчики в тестировании трятят значительную часть своего рабочего времени на автоматизацию больших, сквозных регрессионных тест-кейсов. Именно эти тесты показывают, могут ли все части продукта работать слаженно на благо конечного пользователя. Подавляющее большинство таких тестов пишется на Java с использованием Selenium для управления браузером и хранения логики тест-кейсов. Но этого подхода есть недостатки.

- **Трудности перевода.** Логика теста пишется на одном языке, а выполняемое приложение на другом (Java вместо JavaScript). Разработчики и тестировщики в Google часто жалуются на эту разницу, потому что она значительно замедляет отладку, и не каждый инженер хочет учить дополнительный язык.
- **Место жительства.** Код тестов живет за пределами браузера, поэтому приходится делать дополнительный шаг для сборки и развертывания тестовых бинарных файлов на компьютерах. Централизованная инфраструктура автоматизации тестирования Matrix, к сожалению, не решает проблему полностью.
- **Окружающая среда.** Тестировщик работает в установленной локально среде разработки, отделенной от браузера и настроенной только для тестируемого проекта.
- **Потерянное время.** Тестировщики тратят много времени на постоянное переключение между страницей приложения и средой Eclipse. Они ищут XPath-пути нужных элементов, а потом вручную добавляют их в Java-код. Потом сборка, запуск, проверка работоспособности. Все это требует времени и достаточно утомительно.

- **Ничто не вечно.** Веб-приложения Google часто меняют свою модель DOM. Это значит, что тест-кейсы падают при каждом изменении положения элемента на странице или его атрибутов. Поэтому команды сначала тратят много времени на сопровождение тестов, а потом и вовсе игнорируют полученные результаты из-за обилия ложноположительных срабатываний.

Мы придумали веб-решение этих проблем: Record and Playback Framework (RPF) на основе JavaScript, а еще мы серьезно поработали над хранением сценариев тест-кейсов в облаке. Это решение отлично работает и в Chrome OS, которая не поддерживает выполнение тест-кейсов Selenium или WebDriver.

Чтобы записать тест, просто нажмите Record and Playback в BITE-меню в браузере. На экране появится окно записи, в котором запишутся все операции мышкой в основном окне браузера. Клик правой кнопкой мыши на любом элементе запустит режим проверки, в котором можно проверить конкретную строку, картинку, значение конкретного элемента. Можно даже проверить относительную позицию элемента на странице. Это полезно при работе с YouTube: не всегда известно, где именно будет располагаться видео на домашней странице, но общий макет страницы мы знаем.

Самый главный плюс метода RPF в том, что он избавляет инженера по тестированию от хлопотного просмотра модели DOM приложения и пересчетов путей XPath, когда элементы меняются. Мы вложили много усилий в написание кода, который останавливает тест. Если элемент не найден в процессе воспроизведения, код сделает паузу, чтобы тестировщик выбрал новый элемент, автоматически обновит скрипт и продолжит работу. Еще мы реализовали так называемое «ленивое выполнение»: вместо того чтобы придирчиво проверять, соответствует ли элемент ожидаемому XPath, RPF проверяет все атрибуты элемента HTML, в том числе и его родительские и дочерние элементы в DOM. Во время воспроизведения RPF сначала ищет точное совпадение. Если не находит, начинает искать максимально похожие элементы. Может быть, например, изменился только ID, а все остальное осталось прежним. Точность совпадений поиска настраивается. Если различие в пределах допустимого, тест переходит к следующему шагу и просто записывает предупреждение в логи. Мы надеялись, что этот метод сэкономит много времени разработки.

Первой RPF опробовала команда тестирования Chrome Web Store. RPF успешно отработал в 90% тестовых сценариев. Проблемы возникли только с диалоговыми окнами загрузки файлов, которые по сути — встроенные окна ОС, а не браузера, и с некоторыми функциями Google Checkout: нельзя автоматизировать финансовые сценарии через Web API из-за безопасности. Правда, тестировщиков не сильно захватила идея «ленивого» поиска совпадений или возможность поставить работу на паузу для исправления. Им было проще и быстрее переписать тест с нуля. Все тесты мы поначалу разрабатывали параллельно на два фронта, для WebDriver и для RPF. Оказалось, что RPF в семь раз эффективнее для генерации и сопровождения

тестов, чем Selenium или WebDriver. Показатели могли меняться, но это уже был хороший признак.

BITE использует RPF для записи сценариев при регистрации багов. Для некоторых сайтов BITE автоматически записывает все действия тестировщика, а когда инженер регистрирует баг с помощью BITE, к нему прикрепляется ссылка на генерированный сценарий воспроизведения. Для Google Maps, например, сохраняются все операции поиска и изменения масштаба. Если у разработчика установлен BITE, он может одним кликом запустить воспроизведение и посмотреть, что делал тестировщик, когда нашел баг. Если во время сеанса на сайте баг не заводился, то записанный сценарий самоуничтожается.

Слияние BITE с RPF

Джеймс Арбон

В первые дни тестирования Chrome OS мы обнаружили, что главное качество платформы — безопасность — сильно осложняет тестирование. Тестируемость часто конфликтует с безопасностью, а ведь в Chrome OS очень большой упор сделан именно на безопасность.

В ранних сборках еще была частичная поддержка виртуальных Java-машин (JVM) с ограниченной сетевой функциональностью и поддержкой других базовых библиотек. Так как основные сценарии пользователя основаны на просмотре веб-страниц, мы решили написать несколько тестов с использованием Selenium, чтобы проверить базовую функциональность браузера, и надеялись, что получится просто портировать все уже готовые тесты Selenium для регрессионного тестирования.

Простейшие тесты заработали, но радоваться было рано: мы столкнулись с отсутствием полноценной поддержки Chrome в Selenium и WebDriver. Вернувшись к работе после праздников, мы обнаружили, что из базовой ОС Linux исключили поддержку Java, чтобы повысить уровень безопасности Chrome OS. Конечно, это осложнило выполнение тестов на Java, но мы решили проблему, построив специальную сборку Chrome OS с встроенной поддержкой Java. Это, конечно, было обходное решение, и мы не были им довольны на все сто.

В Google часто говорят, что «дефицит приносит ясность». Это работает в мире тестирования, как нигде больше. Это сработало и для нас в тот момент. Хорошенько оценив ситуацию, мы поняли, что решение было так себе. По сути, мы не тестировали реальный продукт в том виде, в котором им будет пользоваться наш клиент. Мы строили образы Chrome OS, которые содержали Java, артефакты тестирования (jar-файлы), и отключали некоторые средства

безопасности. Посмотрите на фотографию нашей лаборатории автоматизации тестирования ранних версий Chrome OS (рис. 3.41).



Рис. 3.41. Лаборатория тестирования ранних версий Chrome OS

Вскоре нужное решение пришло. Мы вспомнили про проект нашего коллеги По Ху по автоматизации тестирования веб-страниц с использованием JavaScript через расширения Chrome. Это могло сработать. Он назывался Puppet, и это был внутренний API, похожий на WebDriver и работающий только на JavaScript. Правда, из-за межсайтовых ограничений он должен был развертываться вместе с тестируемым веб-приложением. Мы рискнули поместить сценарий Puppet в расширение Chrome, чтобы оно работало для любых сайтов. И — о чудо! — установив только это расширение и сохранив тесты в облаке, мы смогли выполнять браузерные тесты в Chrome OS даже на компьютере Chromebook, только что купленном в магазине. Реализация этой идеи заняла бы у нас больше времени, чем у нас было до выпуска Chrome версии 1, и мы подвинули этот проект в список инструментов, которые нужно разработать к следующей версии.

Кстати, исходная версия BITE называлась Web Test Framework, или WTF, и нам сошло это с рук. Официально считалось, что сокращение происходит от названия, а не наоборот. А вот метод RPF изначально назывался Flux Capacitor¹, так как она позволяла двигаться назад в будущее.

¹ Деталь машины времени из фильма «Назад в будущее». — Примеч. перев.

Ручные и исследовательские тесты в BITE

Мы в Google опробовали уйму способов распределения тестов между инженерами: от недружелюбного TestScribe до электронных таблиц совместного использования, где вручную вводились имена людей напротив тестов, которые они должны провести.

BITE поддерживает подписку тестировщиков на пакеты тестов в Google Test Case Manager для многих продуктов Google. Схема работы проста: когда тест-менеджер хочет начать серию тестов, он нажимает на кнопку на сервере BITE, и тесты доставляются участникам через пользовательский интерфейс BITE. К каждому тесту можно привязать URL-адрес. Если тестировщик принимает запрос на выполнение теста, BITE открывает URL-адрес в браузере и выводит тестовую страницу с последовательностью действий и критериями проверки. Всего одним кликом можно пометить тест как пройденный, после чего автоматически открывается URL-адрес для следующего теста. Если тест не проходит, это записывается в базу, и открывается интерфейс для создания баг-репорта.

Мы успешно опробовали этот метод на краудсорс-тестировщиках. Они выполняли тесты с установленным BITE, причем они и тесты получали через это приложение. Больше не нужно было пристально следить за работой тестировщиков и оперативно распределять между ними тесты — за нас все делал BITE. Те, кто быстро выполнял тесты, автоматически получали новые. Если тестировщик делал перерыв или прекращал работу, его задания просто передавались другому участнику команды. С исследовательским тестированием BITE тоже здорово помог: описание каждого высокоуровневого тура мы оформили как тест, после чего их распределили между тестировщиками, которые уже заводили баги с помощью BITE.

Уровни BITE

Как и любое приложение, внутренние проекты всегда нужно делать расширяемыми. В BITE есть возможность размещения произвольных сценариев и их внедрения в тестируемую страницу. То есть в архитектуре есть несколько логических уровней: один из них, к примеру, позволяет разработчику удалять элементы со страницы в поисках причины бага. Уровни могут включаться и отключаться со специальной консоли. Мы исследуем, какие еще полезные уровни можно добавить. Сейчас, например, мы работаем над включением сценариев от команды безопасности.

BITE был создан как универсальное средство помощи всем тестировщикам. Сначала его фичи были реализованы отдельными расширениями, но команда решила, что целое — это не просто совокупность частей, и мы потратили немало усилий на то, чтобы эффективно свести все воедино в BITE.

Как и в случае с другими экспериментами, команда надеется вскоре открыть доступ к проекту широкому сообществу тестирования.

Проект BITE был переведен на модель открытого кода (подробнее в приложении В). Первым техническим руководителем был Алексис О. Торрес; сейчас проектом руководит Джейсон Стредвик. С ним работают Джо Мухарски, По Ху, Дэниел Дрю, Джуллия Ральф и Ричард Бастаманте, когда им удается выкроить минутку в своих текущих проектах. На момент написания книги некоторые внешние компании внедряли BITE в свою инфраструктуру. Сейчас мы работаем над добавлением поддержки Firefox и Internet Explorer.

Google Test Analytics

Несмотря на то что анализ рисков нужен разработке как воздух, этот процесс часто происходит как попало. Если данным вообще удается покинуть головы участников команды, то часто они просто фиксируются в таблицах. Что в этом плохого?

- У данных нет единой схемы, потому что каждая таблица создается под конкретную ситуацию. Данные нельзя связать между собой, а это очень неудобно, если вы следите за несколькими проектами.
- Простые, но важные вещи, например четырехбалльная шкала оценки и общая схема названий из АСС-анализа, порой теряются при попытках сократить количество полей в таблицах.
- Данные не хранятся централизованно, поэтому недоступны всегда и всем. Командам приходится запрашивать друг у друга информацию устно при каждой необходимости.
- Разработка скриптов, которые связали бы анализ рисков с метриками продукта, обычно обходится дорого, поэтому редко сочетается с таблицами.

Google Test Analytics (GTA) — наша попытка решить эти проблемы. В интерфейс GTA встроены методы АСС-анализа, это простое приложение упрощает ввод данных и работу с рисками. Все данные представлены по одной схеме — менеджеры и директора могут легко получить сводку рисков по всем своим проектам, чтобы перераспределить ресурсы в более опасные области.

Итак, GTA поддерживает модель анализа рисков АСС. Атрибуты и компоненты вводятся в простые формы и формируют таблицы (рис. 3.42 и 3.43), а интерфейс позволяет добавлять возможности в ячейки при планировании тестирования (рис. 3.44). Чтобы добавить риск, нужно просто выбрать частоту и степень воздействия из выпадающих списков для каждой возможности. Все эти значения

сводятся в общую витрину рисков. Итоговый риск для каждой области (рис. 3.45) считается простым усреднением рисков по ней¹.

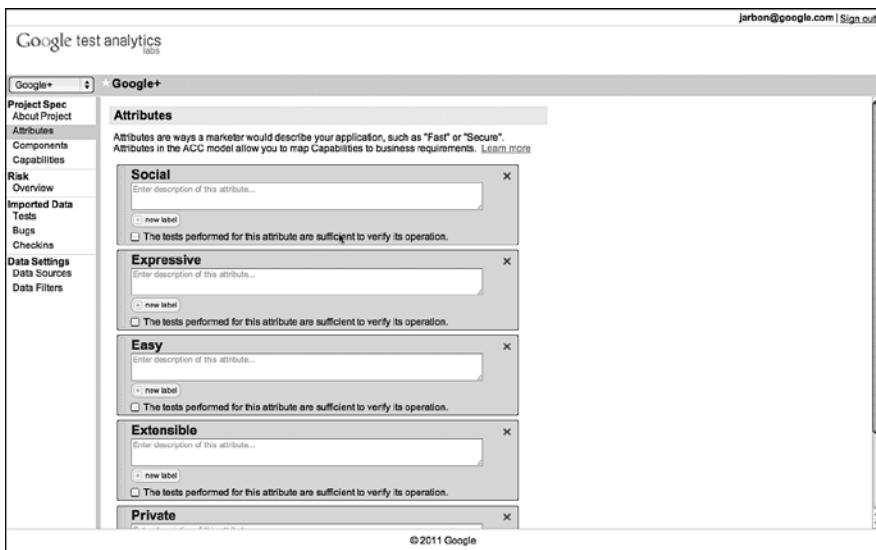


Рис. 3.42. Test Analytics: ввод атрибутов для Google+

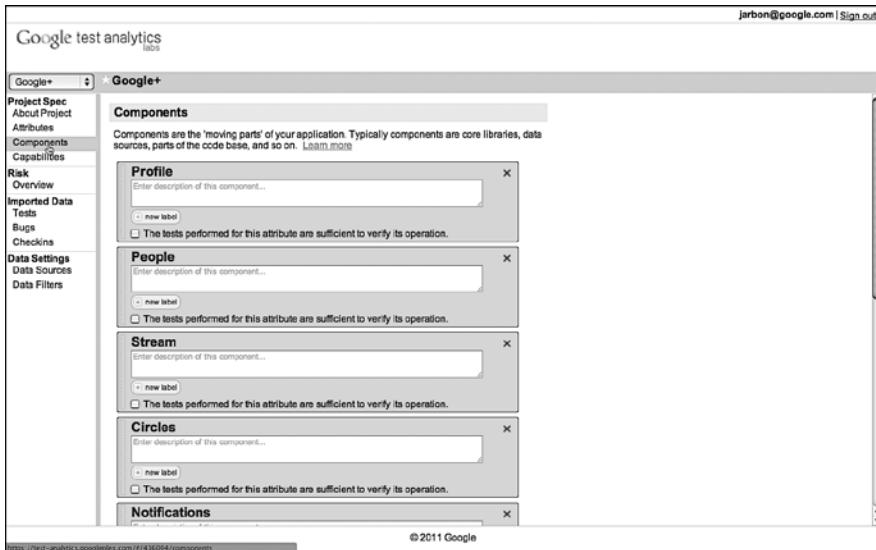


Рис. 3.43. Test Analytics: ввод компонентов для Google+

1 Да, получается, что одна способность высокого риска может потеряться в темном лесу других, менее рискованных. Это редкая ситуация, но мы умышленно создавали очень простой инструмент, помогающий думать, а не полностью работающий за вас.

Capabilities by Attribute and Component

	Social	Expressive	Easy	Extensible	Private
Profile	1	2	1	1	2
People	1	1	1	1	1
Stream	1			1	1
Circles	1	1	2	1	1
Notifications			1	1	1
Hangouts	3	1	3	4	2
Posts		1		1	1
Comments		1		1	1
Photos	1		2	1	1

Hangouts is Easy

Add new capability... Add

Additional users can be added to an existing hangout.

Name: Additional users can be added to an existing hangout.
new label
Description: Enter description of this capability...

© 2011 Google

Рис. 3.44. Test Analytics: ввод возможностей в ячейке. Обратите внимание: вводится количество возможностей на пересечении, а не величина риска

Known Risk

This shows the Total Risk to your application, taking into account any Risk Sources as well as Mitigation Sources that are checked below. [Learn more](#)

Inherent risk Bugs Code churn Test coverage

Risk displayed by Attribute and Component

	Social	Expressive	Easy	Extensible	Private
Profile					
People					
Stream					
Circles					
Notifications					
Hangouts					
Posts					
Comments					
Photos					

© 2011 Google

Рис. 3.45. Test Analytics: карта рисков для Google+

Возможность связать расчет рисков с текущими данными проекта — это *опциональная* экспериментальная возможность GTA. Вы добавляете новые тесты,

пишете новый код, находите новые дефекты, и оценка риска меняется. Как инженеры по тестированию, мы и так всегда держали все эти изменения в голове, эта фича просто позволяет опираться на фактические данные и делать это более системно. Тест-планы, даже основанные на рисках ACC, часто хороши в начале планирования тестирования. Но скоро они могут устареть и покрыться пылью. Хотя в GTA всегда можно внести любые изменения при получении любых новых данных, мы хотим автоматизировать и эту сторону планирования тестирования.

Пока GTA умеет связывать свои данные только с нашими внутренними базами, но мы работаем над избавлением от этой зависимости в будущем. Работая с возможностями в GTA, тестировщики могут ссылаться на секции или запросы в багтрекинговой системе, ветку кода или номер тест-кейса. В Google все используют одни и те же базы, поэтому это работает. Как только метрики багов, кода или тестов меняются, запускаются простые подсчеты и риски пересчитываются. Сейчас несколько команд разработки испытывают этот метод на себе.

Правила расчета, которые мы используем, постоянно меняются, поэтому мы их не будем здесь приводить. По сути, они показывают, как изменилось число багов, строк кода, прошедших и упавших тестов к завершению оценки рисков. Мы учитываем то, что у разных команд разные подходы, ведь некоторые заносят даже мельчайшие баги или по-другому измеряют свой код. Каждый компонент риска масштабируется в рамках условий проекта. Посмотрите примеры связей данных с рисками для Google Sites на рис. 3.46–3.48.

The screenshot shows the Google Test Analytics interface for the 'Sites-Test' project. The left sidebar lists various settings like Project Spec, Risk Overview, and Data Sources. The main area has tabs for 'Add Data Request' and 'Existing Requests'. Under 'Existing Requests', there are two entries: 'Buganizer' and 'Google TCM', each with a 'Save' and 'Cancel' button. The top right corner shows the email 'jarbon@google.com' and a 'Sign out' link. The bottom right corner has a copyright notice: '© 2011 Google'.

Рис. 3.46. Test Analytics: связь источников данных с риском

Google test analytics

Sites-Test ★ Sites-Test

Project Testcases

The following testcases have been uploaded to your Google Test Analytics project. [Learn more](#)

Showing 928 Testcases

Testcase	Attribute	Component	Capability
Layout - Undo [google-testcasemanager.appspot.com]	✗	✗	✗
Layout - Client side caching [google-testcasemanager.appspot.com]	✗	✗	✗
Layout - Gadgets in different layouts [google-testcasemanager.appspot.com]	✗	✗	✗
Page level notifications [google-testcasemanager.appspot.com]	✗	✗	✗
Rename Page: Use special characters to rename a page [google-testcasemanager.appspot.com]	✗	✗	✗
Page with a lot of content [google-testcasemanager.appspot.com]	✗	✗	✗
Insertion of all Gadgets in the web page [google-testcasemanager.appspot.com]	✗	✗	✗
Client side caching - FF browser [google-testcasemanager.appspot.com]	✗	✗	✗
Text/Image - Two column layout [Based on the bug -1365235] [google-testcasemanager.appspot.com]	✗	✗	✗
Messages for not logged in users. [Based on the bug -1686281] [google-testcasemanager.appspot.com]	✗	✗	✗
Create Dashboard page [google-testcasemanager.appspot.com]	✗	✗	✗
Creating a page with the Announcements	✗	✗	✗
No 'Subscribe to postal' link when site is not public [google-testcasemanager.appspot.com]	✗	✗	✗
Owners - All Dasher Admins automatically get ownership rights [google-testcasemanager.appspot.com]	✗	✗	✗
Site Elements - Header [google-testcasemanager.appspot.com]	✗	✗	✗
Site Elements -Configure countdown Date/Year adjustment [google-testcasemanager.appspot.com]	✗	✗	✗

© 2011 Google

Рис. 3.47. Test Analytics: связанные тесты

Google test analytics

Sites-Test ★ Sites-Test

Project Bugs

The following bugs have been uploaded to your Google Test Analytics project. [Learn more](#)

Showing 5482 Bugs

Bug	Attribute	Component	Capability
Add logging to Deferred [v]	✗	✗	✗
jobtok.js reports Dojo Error [v]	✗	✗	✗
Recall of Page with Revision Argument [v]	✗	✗	✗
bindableArray removeAll() not working [v]	✗	✗	✗
listcomponent itemList throws errors upon marshalling [v]	✗	✗	✗
Friendly marshalling errors [v]	✗	✗	✗
I18n problem with ListComponent [v]	✗	✗	✗
can't marshal an object stored in property of type object/json [v]	✗	✗	✗
patchedFlag in transmire.rpc frame is unusable [v]	✗	✗	✗
Allow for external script load call location to be configurable [v]	✗	✗	✗
no_payload_available for posted events [v]	✗	✗	✗
support static methods in omnipresent calls [v]	✗	✗	✗
support loading extra page-specific js on-demand [v]	✗	✗	✗
appfactory.Page.prototype.save needs to use <code>jot.lib.types.castProperty()</code> for main/text	✗	✗	✗
Improve diagnostic messages for omnipresent_constructor_registry errors [v]	✗	✗	✗
Improve Deferred error handling [v]	✗	✗	✗
non-usercontent pages need an option to auto-marshall main/text [v]	✗	✗	✗
is pulled from a <code>obj</code> when <code>property</code> marshalls differently than other <code>js</code> [v]	✗	✗	✗
client-side template rendering should appropriately escape output [v]	✗	✗	✗
supply property type metadata when marshalling so on client-side we can differentiate between string and richtext fields	✗	✗	✗
optimize Page.refresh() [v]	✗	✗	✗
cache Page objects [v]	✗	✗	✗
main/text should not be passed back for system pages [v]	✗	✗	✗

© 2011 Google

Рис. 3.48. Test Analytics: связанные баги

Легко не заметить одну очень важную функцию в GTA: тестировщики могут быстро превратить список возможностей в серию тестов. Команды очень просили

добавить именно эту фичу. Смысл в том, что возможности — это простой список высокоДуровневых тестов, который нужно прогнать перед выпуском программы. Для маленьких команд, которые фокусируются на исследовательском тестировании, например как команда Google Docs, этот список можно легко использовать вместо базы данных тестов.

В GTA используется матрица ACC-анализа, и это дает тестировщикам кардинально новый подход. Обычно тестовые серии или задания на разработку тестов назначаются по *компонентам*. ACC позволяет распределять тестировщиков по *атрибутам*. Наши эксперименты показали, что такой фокус работает лучше всего. Если за тестировщиком закрепить атрибут «быстрый» для всего набора тестов, то он может оценить, насколько быстро работают все интересующие компоненты продукта за одну серию тестов. Так можно выловить медленные компоненты, которые при независимом тестировании могли прикидываться достаточно быстрыми.

Что же со связями и зависимостями между рисками разных проектов? В GTA эта фича пока не реализована. Каждый проект делает свой ACC-анализ и оценивает риски только для своего проекта, без учета других проектов компании. Если кто-то хочет проанализировать риски нескольких продуктов сразу, ему нужно нормировать данные между проектами, чтобы смотреть на них в совокупности. Только то, что ваша маленькая команда работает над внутренним инструментом, не значит, что у вас не может быть максимальных значений рисков. Оставьте относительность тем, кто видит много проектов сразу. Когда оцениваете риск для своего проекта, оценивайте его так, как будто ваш проект — единственный в компании. В нем вполне могут быть часто срабатывающие риски с высокой степенью воздействия.

Сейчас проект GTA используется еще в нескольких компаниях, и мы хотим сделать GTA общедоступным продуктом с открытым кодом. Мы хотим, чтобы другие команды тестирования могли устанавливать у себя свои системы на движке Google App Engine или даже портировать код и разворачивать на других платформах.

GTA делает анализ рисков настолько простым и удобным, что люди действительно им пользуются. Джим Рирдон вырастил GTA с нуля и сейчас поддерживает его в опенсорсе (подробнее об этом мы рассказываем в приложении В). На момент написания книги несколько больших компаний, занимающихся облачным тестированием, хотят интегрировать эту технологию в свои рабочие процессы и инструменты¹. И еще около 200 внешних инженеров записались на использование GTA.

¹ Одной из таких облачных компаний является Salesforce. Фил Валигора из SalesForce.com занимается интеграцией GTA во внутренний инструментарий.

Бесплатное тестирование

Google сокращает время отклика, борясь буквально за каждую миллисекунду, и старается сделать свои системы суперэффективными. И конечно, нам нравится делать свои продукты бесплатными. Наши тестировщики делают то же самое со своими инструментами и процессами. Google просит нас мыслить масштабно, так давайте рискнем и снизим стоимость тестирования почти до нуля!

Почему нам это интересно? Работает простая схема: если тестирование бесплатно, маленькие компании и стартапы смогут позволить себе тестирование. Если тестирование бесплатно, интернет становится лучше, а это интересах пользователей и Google.

Мы представляем, что бесплатное тестирование это:

- почти нулевые затраты;
- мгновенное получение результатов;
- минимум человеческих ресурсов;
- супергибкость. Мы не верим, что всем может подойти одно и то же.

Чтобы задача соответствовала времени и остальным проектам Google, мы решили начать с веб-тестирования, надеясь, что к моменту завершения работы весь мир все равно будет работать в облаке, а драйверы и СОМ останутся в прошлом. Мы знали, что если будем создавать бесплатный продукт, то при любом исходе получим что-то интересное.

Итак, у нас получилась модель, которая здорово сокращает затраты на тестирование и ловко обходит проблемы. В наших лабораториях уже созрели первые плоды такой модели (рис. 3.49). Схема работы такая.

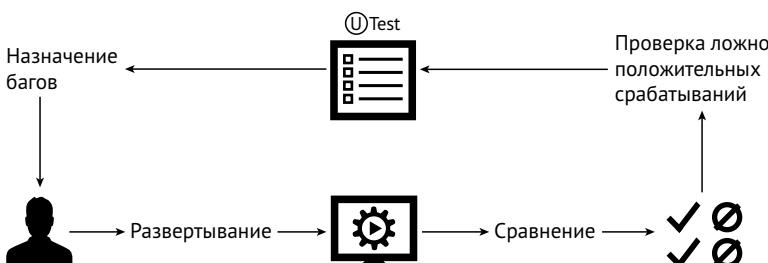
1. **Планирование с GTA.** Основанное на анализе рисков, быстрое и автоматически обновляемое планирование тестирования.
2. **Тестовое покрытие.** Боты проверяют каждую новую версию сайта, индексируют его содержимое, ищут различия. Бот, конечно, не сможет отличить регрессионный баг от новой фичи, но он заметит изменения и сообщит о них человеку.
3. **Оценка багов.** Обнаруженные различия автоматически передаются специалистам для оценки. Итак, регрессия или новая фича? Здесь поможет BITE, который показывает информацию о заведенных багах в контексте этой оценки.
4. **Исследовательское тестирование.** Непрерывное исследовательское тестирование силами краудсорс-тестировщиков и ранних последователей. Они помогут поймать баги, связанные с конфигурациями и такие, которые сможет заметить только человек.

5. **Регистрация багов.** Всего несколько кликов — и баг зарегистрирован, а в отчете можно найти снимки экрана, конкретное место появления и информацию для отладки.
6. **Приоритизация и отладка.** Разработчики и тест-менеджеры почти что в реальном времени получают сводную и очень подробную информацию о багах. Все необходимые данные уже есть, и даже можно в один клик просмотреть, как именно тестировщик нашел баг.
7. **Развертывание новой версии и возвращение к шагу 1.** Повторяем все заново.

Веб-тестирование стремится к автоматизации и работе в духе поисковых систем. Наш метод как раз в духе времени — тестировщику не нужно пробираться через заросли регрессионных тестов, чтобы обнаружить, что это не баг, а новая фича поломала тест. Боты могут работать круглосуточно, а цикл тестирования у них завершается за минуты. Получается, что тесты могут проводиться чаще и регрессии будут обнаруживаться раньше.

Самая приятная часть работы с ботами — сокращение времени между выкачиванием версии продукта и поимкой багов. Итак, если боты и тестировщики из сообщества могут работать круглосуточно, значит разработчики могут оперативно получать данные о последствиях своих изменений в коде. Так как сборка и установка происходят непрерывно, то легко определить, какая именно горстка изменений привнесла баг. К тому же все изменения еще свежи в голове разработчика, и процесс отладки прост и понятен.

Описанный процесс хорошо подойдет для сайтов, но теоретически может быть применим и для клиентских UX-проектов, приложений, построенных на данных, и даже для инфраструктурных проектов. Представьте, что параллельно разворачиваются две версии вашего продукта, и подумайте, как бы выглядели обход и индексирование вашего приложения. Скорее всего, вы найдете много общего с нашей схемой работы, но, впрочем, это уже совсем другая история.



Rис. 3.49. Сквозной процесс бесплатного тестирования

Инновации и эксперименты в тестировании

Джеймс Арбон

Мы в Google за любые эксперименты, поэтому у нас и создается множество инноваций. Ну и куча неудачных экспериментов заодно. Даже если уже есть хорошее решение, мы не запрещаем инженерам пытаться придумать еще лучше. Собственно, переосмысливать, анализировать и улучшать — это их работа.

Когда Джеймс Уиттакер пришел в Google, первым делом он организовал внутреннюю встречу технических специалистов и рассказал, каким он видит будущее тестирования. В его понимании тестирование должно напоминать видеоигры: как и в шутерах от первого лица, на тестируемое приложение должна накладываться полная контекстная информация. Мало кто мог представить, что его доклад на конференции GTAC¹ задаст тон работе на несколько лет вперед. Конечно, на слайдах его идеи смотрелись хорошо, но построение общего решения для всех клиентских приложенийказалось делом очень дорогим и сложным.

Я скептически относился к смелым идеям Джеймса, пока вдруг не осознал, что уже могу почти моментально реализовать их в браузере для веб-приложений с помощью API новых расширений Chrome. Идея захватила меня настолько, что всю следующую неделю я провел в работе над прототипом. Я даже приостановил свою текущую работу и провел выходные в Starbucks, неистово программируя. Сотрудники кофейни интересовались у меня, не ищу ли я работу в интернете, и мне оставалось только постучать по дереву.

Вскоре у меня была рабочая демоверсия с расширением Chrome, которая работала с Python App Engine² и моделировала обращения к базе данных багов. Мне удалось продемонстрировать несколько интересных моментов:

- наложение информации о багах на страницу и даже на ее конкретные элементы;
- наложение тест-кейсов на тестируемую страницу с одной кнопкой для результата теста (рис. 3.50);
- тепловую карту, на которой было видно, чем раньше занимались другие тестировщики и какие значения использовали (рис. 3.51).

¹ Презентация Джеймса Уиттакера на конференции GTAC, посвященная будущему тестирования, доступна на YouTube по адресу http://www.youtube.com/watch?v=Pug_5Tl2UxQ

² App Engine — облачный сервис Google для размещения сайтов и сервисов. Тестировщики часто используют App Engine для инструментов и инфраструктуры, так как App Engine может очень быстро наладить работу приложения и дать возможность пользоваться Google Scale бесплатно. Можно посмотреть на <http://appengine.google.com>. В настоящее время поддерживаются языки Java, Python и Go.

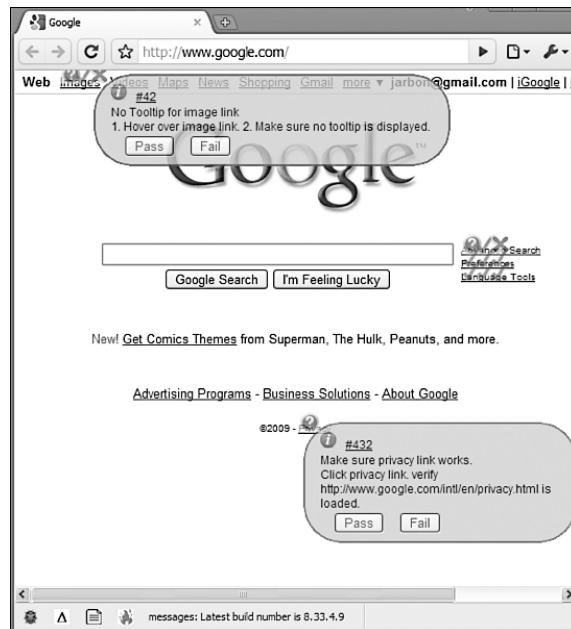


Рис. 3.50. Пользовательский интерфейс тестовых примеров

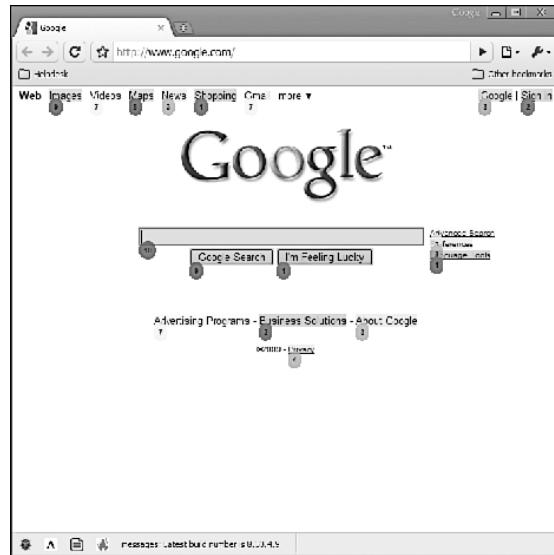


Рис. 3.51. Термовая карта тестового покрытия

Технология заработала на google.com, и я начал пробовать ее на нескольких других сайтах, чтобы убедиться в ее работоспособности. Я назначил короткую неформальную встречу с Джеймсом, чтобы показать результаты своей работы и узнать, что он о них думает. Мы исписали всю доску тем, что потом стало планом для экспериментов, описанных в этой книге. Я отправил Пату Коупленду и Джеймсу письма, в которых сообщил, что теперь буду заниматься этой работой и отчитываться перед Джеймсом. Никаких вопросов не было — изменения были согласованы по почте.

Все следующие эксперименты проводились по той же схеме — каждый инженер отвечал за свое задание и, если хотел, привлекал других. Главная задача управления — добиться того, чтобы работа могла быть применима повторно, была гибкой и ею без проблем могли бы воспользоваться другие люди. Постоянно просите инженеров мыслить масштабно при любой работе!

Культура Google поощряет обмен идеями, поддерживает инициативу снизу, а организационная гибкость дает плодородную почву для экспериментов. Никогда не знаешь, чем это закончится, пока не попробуешь идею на реальных инженерных задачах. Мы понимаем это и даем инженерам возможность ставить любые эксперименты, с условием, что они знают, как оценить их успешность.

Внешние тестировщики

В Google работает много талантливых ребят, но мы отлично понимаем, что и наши возможности имеют пределы. Новые амбициозные проекты появляются без предупреждения, и им часто нужны узкоспециализированные профессионалы для тестирования. Мы не всегда можем быстро переквалифицировать наших людей или нанять новых, продукт нужно выпускать быстрее. Google разрабатывает продукты во многих областях: от гаджетов до корпоративного ПО, от операционных до платежных систем. Задач множество — пересобрать ядро операционной системы, изменить пользовательский интерфейс, проверить, работает ли устройство со всеми телевизорами на рынке, — и везде нужны специалисты. Мы понимаем, что иногда нам нужна помощь со стороны, поэтому обращаемся к внешним компаниям-подрядчикам.

Хорошим примером такой работы служит Chrome OS. Мы с самого начала поняли, что сетевые подключения Wi-Fi и 3G — очень рискованная область, потому что она зависит от производителя устройства. Облачное устройство без связи с интернетом теряет свой смысл как продукт. Более того, все обновления безопасности и версий программных продуктов тоже происходят по Сети, и если возникают проблемы с передачей данных по 3G, наша работа обесценивается. Это не та вещь, которую можно оставить на откуп неспециалистам, пусть и с благими намерениями.

Когда мы только начинали работать с устройствами, своих специалистов с нужным тестовым оборудованием у нас еще не было. Впрочем, даже если бы оно было, правильно его использовать все равно никто не умел. Поэтому тестирование проводили внешние специалисты, которые вручную переключались между 20 современными роутерами Wi-Fi. За пару недель эти ребята сообщили нам о многочисленных проблемах при переключении роутеров и о снижении скорости передачи, если квартира пользователя находится в зоне действия нескольких роутеров¹. Конечно, были и другие проблемы, обнаруженные тестированием, но не такие важные. Мы получили графики, которые свидетельствовали о серьезном падении скорости передачи (примерно как на рис. 3.52 и 3.53).

Мы не ожидали резких скачков интенсивности сигнала, но, как видите на рисунке, они были. Разработчики использовали эти данные для решения проблем во время внутреннего тестирования.

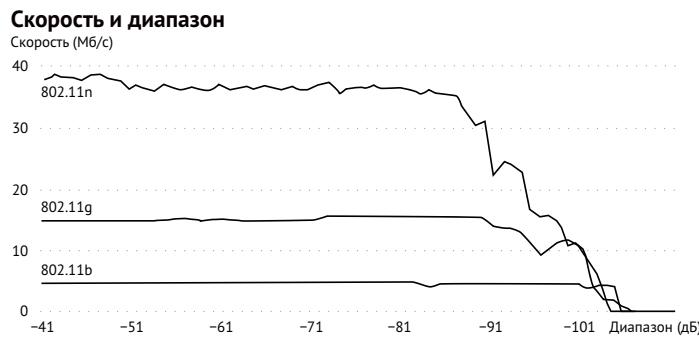


Рис. 3.52. Ожидаемый график зависимости скорости передачи от диапазона

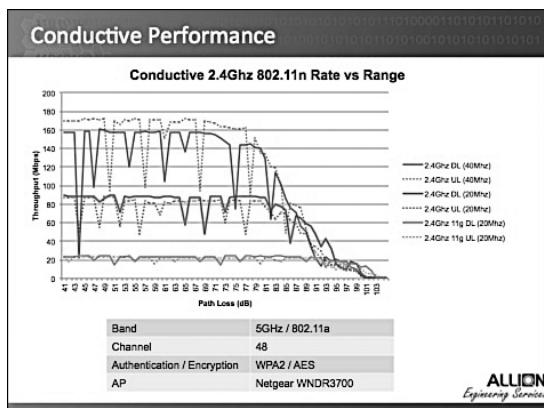


Рис. 3.53. График зависимости скорости передачи от диапазона в ранних прототипах Chrome OS

1 Спасибо Райану Хоупсу и Томасу Флинну из Allion Test Labs за помощь с тестированием и сертификацией оборудования и сетевой поддержки.

Кстати, даже инженеры начального уровня в Google могут привлечь к работе внешних специалистов. Такая способность быстро решать задачи помогает нам быстро выпускать версии наших продуктов. Сейчас у нас есть все необходимое для того, чтобы выполнить работу самим, продолжая то, что начали внешние ребята. Но все же именно то, что мы смогли быстро подключить их экспертизу, стало основной причиной стабильно работающего сетевого соединения в боевых условиях к моменту запуска Chrome OS.

Работа с внешними тестировщиками может не только обнаружить неожиданные дефекты продукта, но и принести столь же неожиданную выгоду. Например, мы попросили их проверить наш список тестов пригодности оборудования. Поставщики оборудования по нашей просьбе отрабатывают эти тесты у себя перед тем, как отправить устройства в Google для дальнейшего тестирования. Такая схема работы избавляет нас от лишних пересылок оборудования туда-сюда. Итак, наши внешние специалисты заметили, что некоторые важные части в тестах были упущены. Ребята здорово помогли нам и отшлифовали наши тест-кейсы в соответствии с принятыми стандартами в этой области. В результате наши первые тесты, переданные крупным производителям компьютеров, стали полными и читались легче. Получается, что выгодно быть скромным и не стесняться обратиться за помощью в тестировании к внешним специалистам.

Интервью с инженером по тестированию Google Docs Линдси Уэбстер

Линдси Уэбстер — инженер по тестированию проекта Google Docs в нью-йоркском офисе Google. В компании ее знают как инженера-практика, способного привести в порядок процедуру тестирования в любой команде разработчиков. Ее подход к работе и умение влиять на команду и качество продукта сделали ее символом инженеров по тестированию в Google.

Недавно авторы пообщались с Линдси, чтобы узнать о ее методах работы.

— Как ты начинаешь работу с новым проектом? Какие первые вопросы ты задаешь? Что ты делаешь в первую очередь?

Линдси: Когда я приступаю к новому проекту, прежде всего я смотрю на него глазами пользователя. Если возможно, я сама начинаю им пользоваться через свой аккаунт и свои данные. Я пытаюсь испытать те же чувства, которые появляются у пользователя при работе с продуктом. Взгляд на программу полностью

меняется, когда вы видите в ней свои личные данные. Освоившись, я начинаю изучать продукт «от» и «до». Если есть проектная документация, то я изучаю ее, а если есть отдельные описания основных фич, я и за них возьмусь. Дайте мне документ, и я его прочитаю.

Как только все документы просмотрены, я начинаю изучать состояние проекта, а конкретно — его состояние качества. Я смотрю на количество багов, как они сгруппированы. Смотрю на типы открытых багов: какие из них открыты дольше всего, какие созданы недавно — и пытаюсь определить соотношение числа найденных и исправленных багов.

— Число багов на разработчика или на всю команду?

Линдси: На всех! Если честно, чтобы достичнуть максимальных результатов, команду надо знать как свои пять пальцев.

Еще я изучаю код в репозитории: ищу подходящие юнит-тесты для каждого класса. Проходят ли эти тесты? Насколько эти тесты содержательны и полны? Есть ли интеграционные или сквозные тесты? А проходят ли они до сих пор? Насколько успешно они проходили раньше? Это самые простые, базовые сценарии или они покрывают и частные случаи? Какие ветки в репозитории меняются чаще всего? Какие из них давно не менялись? Обычно разработчики хорошо документируют все, что они делают по тестированию.

Я исследую любые автоматизированные тесты. Есть ли они вообще? Проходят ли они до сих пор? Я изучаю код этих тестов, чтобы понять, как они проходят сквозь приложение, насколько они полны, хороши ли предположения и условия прохождения или их еще надо доработать. Бывает, что автоматизация покрывает лишь простые тесты. А иногда наборы тестов покрывают сложные пользовательские сценарии, и это хороший знак.

Как только я разобралась со всеми документами, наступает время для командной игры. Я расспрашиваю команду о том, как они общаются и чего ожидают от тестировщиков. Если они используют почтовые рассылки, то я подписываюсь на все, а если есть IRC-канал или другие способы коммуникации — я присоединяюсь к ним.

Спрашивая про ожидания от тестирования, я очень много узнаю о том, что команда разработки не тестирует.

— Нам даже думать тяжело о такой работе. Хорошо, что у нас есть такие тестировщики! Итак, ты расправились с документами и командой, но что-то же осталось на десерт? Само приложение?

Линдси: Точно! Как только я получаю необходимые данные, наступает пора заняться делом. Обычно я разбиваю приложение на функциональные куски. Нестранно, если между ними будет небольшое перекрытие. Моя главная задача — создать не слишком подробную, но и не слишком общую картину, чтобы я могла перечислить все компоненты и фичи.

Когда я разделила приложение на функциональные части, я могу их приоритизировать по очередности тестирования. Какие части приложения я могу назвать самыми рискованными?

Когда с этой частью работы покончено, я возвращаюсь к репозиторию багов и точно так же делю их на смысловые части. Такая работа здорово упростит поиск существующих багов, а это приведет к сокращению дубликатов и подсветит повторные появления тех же самых ошибок.

Далее я прохожусь по своему списку и создаю для каждого компонента пользовательские истории в порядке приоритетов. Для более сложных функций, требующих пошаговых инструкций, я пишу тест-кейсы и связываю их с пользовательскими историями нужного компонента. Я всегда стараюсь прикладывать снимки экранов или видеозаписи или даже ссылаюсь на самые коварные баги в этой области.

Как только у меня есть список тестов, я начинаю искать пробелы в покрытии, заново просматривая баги и приложение. Да, работа тестировщика часто циклична. И эта часть не исключение — я снова перебираю разные виды тестирования, от интеграционного тестирования до проверки нагрузки, и изучаю их покрытие.

После того как фундамент моей работы заложен, я просто стараюсь поддерживать его актуальным: обновляю все изменения в тестах, добавляю документацию для новых фич, обновляю снимки экранов и видео для компонентов, которые поменялись. Хорошо сверять тестовое покрытие с багами, которые пробрались через нашу оборону к пользователям. Это указывает на пробелы в тестовом покрытии.

— Какое место занимает пользователь в твоей работе как тестировщика?

Линдси: Очень важное! Ведь я с самого начала стараюсь стать пользователем тестируемого продукта. Я правда не понимаю, как можно тестировать продукт, если не можешь поставить себя на место пользователя и побить им. Вот почему тестирование — это не просто проверка работоспособности, а еще и полезный источник информации о том, насколько просто им пользоваться и соответствует ли приложение стандартам отрасли.

— Как разработчики оценивают твою работу? Если они недооценивают тестирование, как ты себя ведешь?

Линдси: Разработчики часто недооценивают мой вклад, пока не поработают со мной пару месяцев. Закончив описанный мной этап работы, я встречаюсь с командой и знакомлю их со своим процессом тестирования. Только в личном общении можно донести до разработчиков, что я отношусь к их приложению очень серьезно. Я получаю хорошие отзывы о своей работе, а они убеждаются, что попали в надежные руки.

Как только я показываю результаты своей работы — процесс, который я настроила, изменения, улучшения, которые я добавила, — они говорят сами за себя, и все вопросы о моем присутствии в проекте быстремко отпадают.

Еще один, на первый взгляд парадоксальный, но очень важный момент, который заставляет разработчиков ценить мою работу еще больше. Я открыто и прозрачно отмечаю области, которые я не собираюсь тестировать. Я обосновываю, почему они сами это должны делать. Многие тестировщики избегают подчеркивать, что они не собираются тестировать все, чтобы не показаться менее ценным участниками команды. По моему опыту, это приводит к противоположному эффекту. Разработчики уважают честность.

— Расскажи немного о своем тестировании Google Sites. Как ты подошла к проекту? Какие документы ты создавала и в каком формате? Как подала свои выводы и результаты разработчикам?

Линдси: Тестирование Sites стало настоящим испытанием. Продуктом пользовалось огромное количество людей, продукт был куплен, а не разработан в Google, и к тому же существовал в Google намного дольше остальных.

Я начала работу как пользователь, осваивала продукт самостоятельно, создавая свои сайты. Заодно я связалась с людьми, которые много им пользовались. Например, ассоциация домовладения дома, где я живу, несколько месяцев назад перевела сайт сообщества на Google Sites, и я поинтересовалась, как произошел этот переезд. Документация по этому проекту вовремя не обновлялась, поэтому мне пришлось разбить продукт на блоки и задокументировать все компоненты и подкомпоненты.

То, что продукт разрабатывали не мы, было видно даже по коду. У нас в Google свой подход к написанию, поэтому моя работа немного замедлилась из-за того, что я разбирала чужой почерк. Код находился не там, где я ожидала, его структура была иной. Конечно же, стартапы обычно не пишут много тестов, поэтому в проекте Google Sites, в который превратился купленный Google проект JotSpot, их пришлось дописывать в процессе работы. Мы применили другой подход. Но с такими вещами учишьсяправляться, когда работаешь тестировщиком.

Проект существовал очень давно, и за годы накопилось столько багов, что в репозитории можно было заблудиться. Было бы легче, если бы там была хорошая структура, но разработчики не определяли подкомпоненты, и поэтому перевод всех данных на новую структуру с компонентами занял немало времени.

Я сделала общий сайт (конечно, на базе Google Sites!), где собрала всю документацию для тестирования: пользовательские истории, информацию о среде и команде тестирования, данные по выпускам и т. д. Чтобы быстро планировать тестирование для новой сборки, я использовала простую электронную таблицу со списком всех компонентов и субкомпонентов, упорядоченных по приоритету тестирования.

Закончив работу по реорганизации тестирования, я провела встречу с разработчиками, чтобы дать им полное представление о процессе тестирования. Презентация здорово помогла команде понять масштаб и подводные камни тестирования, а я почувствовала, что моя работа оценена по достоинству.

— А можешь рассказать о самом интересном баге, который ты находила, и как это произошло?

Линдси: Меня всегда забавляло тестирование дат в тех приложениях, в которых надо вводить их вручную. Я люблю использовать даты из далекого прошлого или будущего, и мне часто удается обнаружить странные или даже смешные ошибки вычислений. Я вспоминаю один баг, который каким-то странным образом подсчитывал возраст при вводе в поле даты рождения даты из будущего. Это было довольно забавно. Можете думать обо мне все что угодно, но мне кажется, что баги — это весело!

— Как ты оцениваешь результат своей работы?

Линдси: Важный критерий оценки для меня — количество багов, дошедших до пользователя. Я люблю, когда это число можно округлить до нуля. Еще я серьезно отношусь к тому, что болтают о моих проектах на форумах. Если пользователи ругают продукт за обилие багов или неудобный интерфейс (не сводите глаз с форума пользователей!), я воспринимаю это как сигнал к тому, что мне нужно активнее участвовать в проекте. Проект, кстати, может тащить на себе груз старых багов, которые так и не были исправлены. Поэтому, оценивая свою работу в проекте, я еще учитываю, сколько старых багов продолжает портить жизнь пользователям сегодня. Я стараюсь выводить такие дефекты на первый план, обосновывая повышение приоритета солидным возрастом бага.

— Как ты понимаешь, когда тестирование пора завершать?

Линдси: Трудно сказать. Когда ты пишешь новые версии, то дата выпуска есть финальная черта проекта. А если появляется новая версия браузера или устройства, на которых открывается наше приложение, даже если оно сейчас активно не разрабатывается, то это всегда серьезная причина продолжить тестирование. О'кей, если вы уверены, что если какие-то баги и остались, то они находятся в компонентах, фичах, браузерах, устройствах, которыми пользуются очень редко, а значит, не сильно помешают пользователям, то вы можете приостановить тестирование. Руководствуйтесь приоритетами фич и окружений, которые вы поддерживаете.

— А как исправляются баги?

Линдси: Важная часть моей работы — подталкивать разработчиков исправлять баги. Я постоянно пытаюсь оторвать разработчиков от работы над новыми фичами и отправить их исправлять баги. Чем больше жалоб на баг от пользователей мне удается собрать, тем легче мне убедить разработчика, что нужно уничтожить дефект, вместо того чтобы прикручивать новую функцию. У корпоративных продуктов в Google, таких как Sites, есть специальные службы поддержки клиентов. Я обязательно работаю с ними, чтобы быть в курсе, на что чаще всего жалуются клиенты.

— Если бы ты могла взмахнуть волшебной палочкой и изменить один аспект твоей работы, что бы это было?

Линдси: Если я скажу «все», то мое желание не исполнится? Хорошо, если бы могла, я бы создала простые, примитивные тест-кейсы или пользовательские сценарии, которые бы не надо было документировать, потому что они чудесным образом стали бы известны каждому тестировщику. Скажем, операции создание-чтение-обновление-удаление используются везде, поэтому каждый раз описывать их для каждой новой фичи утомительно. Я думаю, переход на высокоуровневые пользовательские истории вместо формализованных тест-кейсов частично решил проблему, но я бы все равно предпочла полностью избавиться от нее.

— Как твоя работа влияет на решение о выпуске продукта?

Линдси: Я решаю, может ли фича пройти в сборку для релиза, помня об интересах пользователя. К счастью, мои команды обычно соглашаются. Я не блокирую выпуск продукта, если для этого нет веских причин. Для меня важно сохранять доверие команды, чтобы люди знали: если я настаиваю, что нужно блокировать, значит, все совсем плохо.

— Что тебе больше всего нравится или не нравится в твоей работе?

Линдси: Мои навыки дают мне профессиональную гибкость, и мне это нравится. Я — технический специалист, но моя работа — думать о пользователе. Назовите проект, в котором не нужны такие люди! Я могу быть полезной в проекте любого типа. При запуске продукта или новой фичи команды часто нервничают, а мое присутствие и работа приносят им спокойствие и уверенность. Я чувствую, что приношу пользу и позитив.

— Чем тестирование в Google отличается от других компаний?

Линдси: Независимостью. Я свободна в выборе проектов, над которыми хочу работать как в основное время, так и в «двадцатипроцентное», — это такая концепция в Google, позволяющая нам проводить один день в неделю, или 20% общего времени, работая над любым проектом. Это расширяет мой кругозор и поддерживает мой энтузиазм, когда кажется, что работа превратилась в сплошной «День сурка».

— Как разработчики в тестировании относятся к твоей работе?

Линдси: Разработчики в тестировании часто не считают важной работу с репозиторием багов и тестирование каждой версии, пока сами не увидят, как это влияет на продукт. Даже если они думают, что автоматизация покрывает все тестовые сценарии (да, как же!), кто-то должен заниматься исследовательским тестированием для разработки новых тест-кейсов. А еще этот кто-то должен отслеживать все баги, найденные автотестами, сравнивать их с более старыми и менее серьез-

ными багами и отзывами от пользователей. Так как я поднимаю эти вопросы, то разработчики в тестировании обычно понимают ценность изменений, которые я приношу в проект. Если и есть такие, кто не очень уважительно относится к моей работе, это те, с кем мы еще не поработали. После совместной работы отношение разработчиков всегда быстро меняется.

— Как организовано твое взаимодействие с разработчиками в тестировании?

Линдси: Я определяю стратегию тестирования для всех, включая разработчиков в тестировании. Если разработчики в тестировании не знают, с чего начать программирование тестов или инструментов, я показываю им документ с приоритетами, где написано, что нужно тестировать в первую очередь, и подкрепляю информацией о багах. Еще я могу дать им обратную связь, показать на реальных данных, насколько их решение действительно предотвращает баги. Так что мое взаимодействие в основном построено на организации и обратной связи, которую я привношу в работу разработчиков в тестировании.

Интервью с инженером по тестированию YouTube Эппл Чоу

Эппл Чоу — инженер по тестированию Google Offers. До этого проекта она руководила тестированием YouTube в офисе Google в Сан-Франциско. Эппл любит сложные задачи и всегда стремится применить к ним новейшие инструменты и методы тестирования.

Недавно мы побеседовали с Эппл и попросили ее рассказать о тестировании YouTube.

— Эппл, что привело тебя в Google? Такое имя наверняка наводило на мысли о работе в другой компании?

Эппл: Ха! Адрес apple@apple.com выглядит очень аппетитно! Но я выбрала Google из-за разнообразия продуктов и возможности поработать с исключительными специалистами. Мне нравится менять проекты и работать на разных флангах, поэтому я решила, что Google — подходящее для меня место. Я могу влиять на работу миллионов пользователей в самых разных областях. Каждый день ставят новые задачи, и мне никогда не бывает скучно. Конечно, бесплатный массаж — тоже плюс.

— Расскажи о процессе собеседования на роли тестировщика и разработчика в тестировании.

Эппл: Google ищет разносторонних людей, которые могут учиться, расти и справляться с любыми задачами. По-моему, этот подход работает при найме инженеров по тестированию, разработчиков в тестировании и разработчиков. Во многих компаниях собеседования проводят на конкретные роли, и люди, которые проводят собеседование, потом становятся вашими коллегами. В Google другой подход к найму сотрудников — собеседования проводят люди из разных команд, поэтому мы получаем много точек зрения. Я думаю, что так мы находим людей, способных вписаться в любую нашу команду. Это важно, ведь в Google можно легко перемещаться с проекта на проект, работать в разных направлениях и менять команды. Вопрос универсальности сотрудника очень важен.

— Ты работала во многих технологических компаниях. Что оказалось самым неожиданным в организации тестирования в Google?

Эппл: Здесь многое устроено иначе. Может, я не очень объективна, потому что люблю Google, но я бы сказала, что наши тестировщики и разработчики в тестировании более подкованы технически, чем специалисты других компаний. Везде, где я работала, у нас были отдельные команды для автоматизированного и ручного тестирования. Понятно, что разработчики в тестировании в Google сами пишут код, это их работа. Но найти здесь инженера по тестированию, который не умеет программировать, сложно. Такие навыки помогают нам быть более ценными на ранних стадиях, когда до сквозного тестирования еще далеко и надо заниматься юнит-тестированием. Я думаю, что наши технические навыки объясняют наши успехи.

Еще одно отличие Google в тестировании — огромный объем автоматизации. Большинство автотестов уже проходят к тому моменту, когда на проект приходят инженеры по тестированию. Благодаря этому тестировщики получают в работу код очень высокого качества.

Третье отличие — инструменты, которыми мы пользуемся, созданы внутри компании. Мы почти не работаем с коммерческими программами. В нашей культуре очень высоко ценится разработка собственных инструментов, и многие тратят свое «двадцатипроцентное» время на такие проекты. Наши изобретения помогают нам с легкостью преодолевать рутинные и утомительные части тестирования и фокусировать усилия людей на сложных задачах, где действительно нужно участие человека.

Есть еще один важный момент: культура «разработчики-отвечают-за-качество». Разработка строится вокруг тестирования, и в этом мы можем легко положиться на разработчиков. Мы все в Google играем на стороне качества продукта, и любой инженер может протестировать любой код с любого компьютера. Поэтому вся работа идет быстро.

— Отличия понятны, а чем тестирование в Google схоже с тестированием в других компаниях?

Эппл: Функциональность приложения, которую трудно автоматизировать, создает нам столько же проблем с тестированием, сколько и любой другой компании. Когда мы в спешке доделываем новые фичи для очередного выпуска, мы тестируем код не так тщательно, как хотелось бы. Идеальных компаний и совершенных продуктов не бывает.

— За какие функциональные области ты отвечала, работая инженером по тестированию для YouTube?

Эппл: Я работала с разными командами и участвовала в выпуске многих новых фич YouTube. Предмет моей особой гордости — запуск новой страницы просмотра роликов, которую мы полностью переработали. Это одна из самых часто просматриваемых страниц в интернете! Другой запоминающийся проект — наше партнерство с Vevo, новым порталом музыкального контента, для которого YouTube обеспечивает видеохостинг и стриминг. Этот проект — совместное творение Sony Music Entertainment и Universal Music Group. В день запуска на нем уже было 14 000 видеороликов. За три месяца после запуска видеоролики Vevo на YouTube собрали больше 14 миллионов просмотров. Еще я координировала работу по тестированию переписанной версии видеопроигрывателя YouTube на базе Flash, когда мы переходили с ActionScript 2 на ActionScript 3, и запуск новых страниц Channel и Branded Partner.

— Каково это — рулить тестированием в Google?

Эппл: Это значит координировать работу не только в своей команде и проекте, но и не забывать присматривать за всеми продуктами, на которые может повлиять наша работа. Например, в проекте Vevo мы постоянно держали в голове проигрыватель YouTube, брендированный компонент просмотра, иерархию каналов, распределение трафика и т. д. Нужно уметь видеть лес за деревьями.

— Как ты адаптировала исследовательское тестирование для YouTube?

Эппл: YouTube — это продукт, предельно ориентированный на пользователя. Он настолько визуальный, что исследовательское тестирование выходит на первый план, и мы стараемся, чтобы его было как можно больше.

— Как тестировщики YouTube восприняли идею исследовательского тестирования?

Эппл: О, она здорово их взбодрила! Все-таки тестировщикам нравится искать баги. В исследовательском тестировании ребятам нужно немного побывать пользователем и его глазами пройти весь тур, его руками попытаться сломать продукт. Поэтому к написанию новых тестов надо подходить творчески, — работа стано-

вится интереснее, удается найти такие баги, которые обычно или пропускают, или тестируют долго и нудно, пока не наткнутся на них.

— Ты упомянула о турах. Джеймс подтолкнул тебя применить практики из его книги?

Эппл: Когда Джеймс пришел в Google, книга только вышла. Он провел пару семинаров и несколько раз встречался с нами. Но он работает в Сиэтле, а мы в Калифорнии, поэтому мы общаемся не так часто. Мы просто взяли туры из книги и попробовали их. Одни сработали, другие — нет, но мы быстро разобрались, какие лучше подходят для нашего продукта.

— Какие именно работали? Можешь назвать?

Эппл: «Денежный тур» с фокусом на фичи, связанные с деньгами (для YouTube это Ads и другие части, завязанные на партнеров), конечно, играл важную роль в каждом выпуске. «Тур по достопримечательностям» с фокусом на важнейшие функции системы. «Тур по неблагополучному району» с фокусом на области, в которых раньше находили баги, и области, к которым относились последние баг-репорты. Эти три оказались самыми эффективными в выявлении багов. Невероятно полезным оказалось рассмотреть заведенные друг другом баги и обсудить, как мы их искали. Концепция туров помогла нам выстроить стратегию исследовательского тестирования. Мы постоянно подшучивали над «антисоциальным туром» (ввод наименее вероятных данных при каждом удобном случае), «туром навязчивого невроза» (повторение одного действия) и «ленивым туром» (ввод минимума входных данных и выбор значений по умолчанию везде, где только можно). Туры не только помогли нам направлять наше тестирование, но и сплотили команду.

— Насколько мы знаем, вы много тестируете YouTube с помощью Selenium. Что тебе нравится и не нравится в автоматизации с использованием Selenium?

Эппл: Нравится простота API, то, что можно писать код тестов на любимом языке программирования, будь то Python, Java или Ruby, возможность прямого вызова кода JavaScript из приложения — потрясающая, суперполезная функция.

Не нравится то, что технология так и осталась браузерным тестированием. Она медленно работает, нужно встраивать точки входа в API, а тесты работают далеко от тестируемого кода. Selenium помогает при автоматизации сценариев, которые человеку особенно трудно проверять, например при вызовах к рекламной подсистеме. У нас есть тесты, которые запускают видеоролики и перехватывают вызовы Ad с помощью Banana Proxy (внутренняя программа проверки безопасности веб-приложений, которая логирует запросы и ответы HTTP). В двух словах: мы перехватываем запросы из браузера и направляем их на Banana Proxy, который их логирует и передает в Selenium. Так мы можем убедиться в том, что исходящие

запросы содержат правильные URL-параметры, а содержимое входящих ответов соответствует ожиданиям. Тесты пользовательского интерфейса работают медленно, менее надежны и дороже в сопровождении. Мы пришли к выводу, что следует оставить несколько больших высокоуровневых смок-тестов для проверки сквозных интеграционных сценариев, а остальные тесты должны быть как можно меньшего размера.

— Большая часть контента и пользовательский интерфейс YouTube работают на Flash. Как вы это тестируете? У вас есть волшебные приемы, позволяющие делать это через Selenium?

Эппл: Никакой магии, к сожалению. Если не считать огромное количество про-деланной работы волшебством. Наши JavaScript API торчат наружу, и мы можем использовать Selenium для их тестирования. Еще есть программа определения различий *pdif*, которая помогает в тестировании отображения превью, определения границ экрана и прочего. Мы делаем серьезную промежуточную обработку потока HTTP для прослушивания трафика, чтобы больше знать об изменениях на странице. Фреймворки As3Unit и FlexUnit помогают нам проверять различные состояния приложения и сравнивать графические изображения. Я бы хотела назвать это магией, но чтобы все это стало реальным, написано очень много кода.

— Какой самый серьезный баг нашла и не пустила к пользователям твоя команда?

Эппл: Самые серьезные баги обычно не так интересны. Был забавный баг в CSS, из-за которого браузер IE вылетал. До этого мы еще никогда не видели, чтобы баги CSS приводили к аварийному завершению браузера.

Нам запомнился один коварный баг, который поймали во время запуска новой страницы просмотра в 2010 году. Когда пользователь в IE7 выводил курсор за пределы области воспроизведения, проигрыватель через какое-то время зависал. Это интересный случай, потому что пользователи столкнулись бы с этим, только если бы долго смотрели один видеоролик и двигали курсором. Тогда все постепенно начинало тормозить, пока проигрыватель не зависал окончательно. Оказалось, что проблемы возникали из-за обработчиков событий и ресурсов, которые не освобождались, многократно выполняя одни и те же вычисления. Если пользователь смотрел короткие ролики или сидел спокойно и не двигал мышкой, баг не проявлялся.

— Расскажи о самом успешном и самом неуспешном моментах тестирования YouTube.

Эппл: Самым успешным был инструмент для получения и проверки проблемных URL-адресов. Эти простые тесты очень сильно помогли нам быстро отлавливать критические баги. Мы добавили фичу, которая упрощала отладку тем, что выда-

вала данные трассировки стека. Разработчикам стало проще выявлять причины проблем и исправлять их. Этот инструмент стал нашим первым флангом защиты при развертывании и сэкономил тестировщикам очень много времени. После небольшой доработки мы настроили его для самых популярных URL-адресов из наших логов и для добавочного списка URL-адресов, отобранных вручную. Это принесло много пользы.

Пожалуй, наименее успешным аспектом было то, что мы полагались на ручное тестирование во время наших еженедельных авралов. Учитывая то, что времени для тестирование мало, код замораживается и выкатывается на бой в тот же день, многие изменения в пользовательском интерфейсе очень трудно автоматизировать, ручное тестирование все еще играет важную роль в нашем процессе выпуска. Это серьезная проблема, и я хотела бы найти для нее достойное решение.

— Сайт YouTube управляетя данными, а большая часть контента определяется алгоритмами. Как вы следите за тем, чтобы правильные видеоролики отображались вовремя и в нужном месте? Твоя команда проверяет качество видеоматериала? Если проверяет, то как вы это делаете?

Эппл: Мы анализируем, сколько раз и какие видеоролики просматриваются, их связи друг с другом и много других переменных. Мы изучаем, как справляется буфер с данными, ошибки кэша, и на основании полученных данных оптимизируем свою глобальную инфраструктуру.

У нас есть юнит-тесты для проверки качества видео. После моего прихода наша команда написала инструмент AS3 Player Helper для углубленного тестирования этого аспекта. Приложение распространяется с открытым кодом¹. Оно построено на тестах FlexUnit, которые используют встроенный проигрыватель YouTube, воспроизводят им тестовые видеоролики и выдают информацию о состоянии проигрывателя и свойствах ролика. В тестовые ролики зашиты большие штрихкоды, они помечают кадры и отрезки времени. Их легко распознать даже после сжатия и потери качества. Еще один прием, чтобы оценить состояние видео, — это делать снимки видеокадров и анализировать их. Мы проверяем правильность пропорций и кадрирования, ищем деформации, искажения цвета, пустые кадры, белые экраны, проблемы синхронизации и т. д. Так эти проблемы оказываются в наших баг-репортах.

— Какой совет ты бы дала другим тестировщикам веб, приложений на базе Flash и веб-служб, управляемых данными?

Эппл: Работая с тестовыми фреймворками или тест-кейсами, делайте их простыми. Изменяйте архитектуру итеративно вместе с развитием проекта. Не пытайтесь сразу решить все проблемы. Смело выбрасывайте то, что вам не подходит.

¹ Исходный код AS3 Player Helper доступен по адресу <http://code.google.com/p/youtube-as3-player-helper/source/checkout>

Если тесты или инструменты слишком сложны в сопровождении, выбросьте их и постройте более гибкие. Следите за тем, сколько вы тратите на сопровождение и диагностику тестов. Соблюдайте правило 70-20-10: 70% малых юнит-тестов, которые проверяют один класс или функцию, 20% средних тестов, которые проверяют интеграцию одного или нескольких модулей приложений, и 10% больших тестов (тех, что обычно называют системными или сквозными), работающих на верхнем уровне и проверяющих работу приложения в целом.

А дальше — назначайте приоритеты и ищите простые средства автоматизации с наибольшим эффектом. Всегда помните, что автоматизация не решит всех проблем, особенно это касается интерфейсных проектов или устройств. Всегда с умом подходите к исследовательскому тестированию, не забывайте отслеживать тестовые данные.

— Скажи честно, тестирование для YouTube — шикарная работа? Целый день смотреть видеоролики с котиками...

Эппл: Да, мы любим повеселиться, не буду врать. У нас как-то была первоапрельская шутка, когда мы выводили все заголовки видеороликов перевернутыми. Я просматриваю много интересного контента, *и это моя работа!* И даже после всего этого я все еще смеюсь над видеороликами с котиками!

Тест-менеджер

Итак, мы выяснили, что в Google есть инженеры по тестированию и разработчики в тестировании, которые играют на стороне пользователей и разработчиков соответственно. А сейчас мы поговорим о третьей роли, которая связывает и координирует их работу. Речь идет о тест-менеджере. Это в первую очередь инженер, которые вносит технический вклад в проект. В то же время именно через него взаимодействуют все команды, от разработчиков и выпускающих менеджеров до менеджеров продукта и технических писателей. Плюс ко всему тест-менеджер обладает управленческими навыками и помогает своим подчиненным в профессиональном росте. Пожалуй, это самая сложная должность в Google.

Кто такой тест-менеджер

Работа по тестированию проектов в Google на самом деле не выглядит так идеально, как описанная в этой книге ситуация, где каждый инженер занят только своим делом. На помощь приходит менеджер, который координирует работу тестировщиков и разработчиков в тестировании, — тест-менеджер. Немного о его месте в иерархии Google: он подчиняется директору по тестированию, у которого может быть несколько таких подчиненных¹, а Патрик Коупленд руководит всеми директорами по тестированию.

¹ Когда мы писали книгу, в Google было шесть директоров по тестированию, у каждого в подчинении находилась небольшая группа тест-менеджеров. Последние обычно руководят рядовыми сотрудниками, но ведущие инженеры часто подчиняются непосредственно своему директору. Такая иерархия должна упрощать совместную работу. Кроме того, многие директора Google (а может, и все) иногда выполняют задачи как рядовые участники проектов.

На роль тест-менеджера мы редко нанимаем людей снаружи, чаще стараемся вырастить их внутри компании. Сотрудники, пришедшие извне, обычно (но не всегда) стартуют с позиции без подчиненных. Даже Джеймс Уиттакер, который был нанят на позицию директора, почти три месяца не имел подчиненных.

Большая часть тех, кто сейчас руководит тестированием, выросли из тестировщиков, и это неудивительно, учитывая, насколько у них широкий спектр задач. Инженер по тестированию управляет задачами по тестированию, держа в фокусе всю широкую картину проекта. Тут рукой подать до управления людьми. Тестировщик понимает большую часть функциональности продукта и общается с гораздо большим количеством инженеров, чем, к примеру, разработчик в тестировании. Однако успешный инженер вовсе не обязательно станет успешным тест-менеджером. В Google любой успех – дело коллективное. Мы все вместе работаем над тем, чтобы выбрать правильных менеджеров и помочь им добиться успеха.

Итак, наш первый совет — *знайте свой продукт*. Тест-менеджер должен быть готов ответить на любые вопросы по использованию своего продукта. Допустим, вы — тест-менеджер Chrome. Если вас спросят, как установить расширение, сменить оформление браузера, настроить синхронизацию, изменить настройки прокси-сервера, просмотреть DOM, найти хранилище cookie, поставить новую версию, то ваши ответы должны отскакивать от зубов. Хороший руководитель знает о своем продукте все — от пользовательского интерфейса до внутренних подробностей работы данных центров.

Однажды я спросил тест-менеджера Gmail, почему моя почта медленно загружается, и я тут же узнал, как работают отдельные части сервера Gmail, и о том, что на этой неделе была проблема с удаленным данными центром. Объяснение было даже слишком подробным, но я понял, что этот человек точно знает, как функционирует Gmail, и владеет самыми свежими данными о его производительности. Это именно то, чего мы ожидаем от всех тест-менеджеров: знать о своем продукте больше всех участников проекта.

Второй совет — *знайте ваших людей*. Конечно, тест-менеджер — эксперт своего продукта и отлично понимает работу, которую нужно выполнять, но все-таки ее непосредственно выполняют инженеры по тестированию и разработчики в тестировании. Чтобы работа выполнялась быстро и эффективно, невероятно важно хорошо знать своих людей и их навыки.

У нас в Google работают очень умные ребята, но их количество не бесконечно. Каждый тест-менеджер, приходящий извне, жалуется, что в его проекте недостаточно людей. Мы лишь дружно улыбаемся в ответ. Мы знаем это и не собираемся ничего менять. Хорошо зная своих людей и их навыки, руководитель может маленькой командой добиться тех же результатов, что и большим количеством людей. В этом сила тест-менеджера.

Мы уже писали, что дефицит приносит ясность и умножает чувство ответственности у всех участников проекта. Представьте, что вы растите ребенка и у вас семь

нянек: одна кормит, другая меняет подгузники, третья развлекает малыша и т. д. Никто из них не сможет дать ребенку больше, чем один любящий, пусть и перегруженный родитель. Когда ресурсов не хватает, приходится оптимизировать весь процесс. Вы быстро находите свои ошибки и учитесь не повторять их. Вы создаете график кормления, размещаете запасные подгузники поближе, чтобы облегчить себе работу.

В тестировании программных продуктов в Google работает такая же схема. Руководитель не может просто бросить всех людей на задачу, поэтому все действия оптимизируются. Автоматизация, не приносящая пользы, уничтожается. Тесты, не выявляющие регрессию, не пишутся. Если разработчики требуют от тестировщиков определенной деятельности, они должны сами в ней участвовать. Люди не придумывают себе работу, чтобы просто не сидеть без дела. Не нужно делать то, что сейчас не принесет ценности.

Задача руководителя — поставить всех на свои места. Если он хорошо знает свой продукт, то легко определит приоритетное направление и те части, которые должны получить нужное покрытие в первую очередь. Если менеджер хорошо знает команду, то он сможет правильно распределить людей по областям работы для максимальной эффективности. Конечно, какие-то задачи останутся невыполнеными. Но если все сделано правильно, это будут низкоприоритетные задачи или задачи достаточно простые, чтобы их можно было доверить внешнему подрядчику, краудсорсерам или внутренним пользователям.

Конечно, руководитель тоже может принять неправильное решение, правда, из-за важности его роли любая ошибка будет дорого стоить. К счастью, у нас сплоченное сообщество тест-менеджеров, его участники хорошо знают друг друга и регулярно обмениваются полезным опытом, повышая общий уровень знаний. Это еще одно преимущество дефицита ресурсов — небольшое количество людей, которые на «ты» друг с другом и могут регулярно встречаться.

Жонглирование людьми и дирижирование проектами

Фирменная особенность инженерной работы в Google — возможность поменять проект. Как правило, у сотрудника появляется такая возможность примерно раз в 18 месяцев. Конечно, мы не заставляем людей это делать. Если инженеру нравятся мобильные операционные системы, никто не будет кидать его бороздить просторы YouTube. Такая мобильность позволяет попробовать разные проекты. Многие хотят оставаться на одном проекте годами или в течение всей своей карьеры, другие предпочитают узнать обо всем, что делает Google. Мы просто хотим, чтобы у сотрудников был выбор.

Такая особенность работы на руку тест-менеджеру. Фактически получается, что можно в любое время нанять к себе сотрудника с опытом в разных областях. Представьте, что ваш проект — Google Maps, и вы привлекаете к работе ребят, которые работали с Chrome и Google Docs! То есть вы всегда можете пополнить свою команду инженерами с необходимым опытом и свежим взглядом на ситуацию.

Конечно, есть и обратная сторона такой подвижности кадров — можно потерять сотрудника в любой момент. Руководитель должен позаботиться о том, чтобы проект не зависел от конкретных людей. Нельзя просто пользоваться звездным тестирующим. Важно, чтобы эта звездность воплотилась в инструмент или другой метод, который смогут потом использовать другие инженеры, чтобы засветиться в том же созвездии.

Тест-менеджер управляет процессом *распределения*. Он может размещать вакансии в специальном веб-приложении, которое просматривают инженеры. Переход не требует формального разрешения от текущего или будущего менеджера, и любой инженер после полутора лет работы в Google может свободно сменить проект. Конечно, скорость такого перехода согласовывается, так как она не должна создавать помех дате выпуска или ключевым точкам плана проекта, но с разногласиями по поводу подобных переходов мы еще не сталкивались¹.

Новички Google распределяются с помощью того же веб-приложения. Тест-менеджер может просмотреть резюме и результаты интервью нового сотрудника, а потом оставить на него запрос. В периоды активного найма обычно есть несколько кандидатов, которые распределяются между проектами. Конечно, существует здоровая конкуренция и менеджер должен отстоять свое право забрать кандидата в свой проект на собрании директоров по тестируанию. На таких собраниях распределение сотрудников решается голосованием, а при равном количестве голосов Патрик Коупленд (или назначенный им человек) решает, кому достанется приз, то есть инженер.

Основные приоритеты распределения:

- Навыки новичка соответствуют специфике проекта. Мы хотим, чтобы у сотрудника была возможность добиться успеха.
- Пожелания нового сотрудника. Обретение работы мечты может сделать его счастливым, а значит, и производительным инженером.
- Потребности проекта. Стратегически или экономически важные проекты иногда получают более высокий приоритет.
- Прошлые распределения. Если проект давно не получал новых сотрудников, возможно, ему стоит отдать предпочтение.

¹ В этом может помочь концепция «20%», о которой мы говорили ранее. Переходя из проекта А в проект Б, инженер занимается проектом Б в свое «двадцатипроцентное» время около квартала, а в следующем квартале меняет пропорции: 80% времени отводится проекту В, а 20% — проекту А.

К распределению не стоит относиться слишком серьезно. Если руководитель не получит сотрудника на этой неделе, он попробует на следующей. С другой стороны, если распределение прошло неудачно, то новичок может поменять свое место, так как переводы осуществляются легко.

Тест-менеджер заботится о получении новых задач. По мере роста его опыта и репутации ему может быть поручено несколько проектов, а его подчиненными могут стать не только инженеры, но и начинающие тест-менеджеры.

Назначение тест-менеджера на проект может проходить двумя способами. Первый: проектная команда проводит презентацию проекта для тест-менеджера в надежде, что он согласится сформировать для проекта команду тестирования. Второй: самый большой начальник Патрик Коупленд может сам назначить тест-менеджера на стратегически важные проекты.

Когда тест-менеджер сам выбирает проекты, ему стоит избегать команд «с душком». Если команда не желает вносить равноправный вклад в качество, пусть они решают свои проблемы с тестированием как хотят. Если кто-то не желает писать малые тесты и обеспечивать покрытие на модульном уровне, не стоит присоединяться к ним и мешать людям копать себе могилу.

Следует обходить стороной проекты с низкой вероятностью успеха или проекты настолько простые, что и разработчики могут выполнить функции тестировщиков. Нет ни одной причины, по которой тест-менеджер необходим в таких проектах.

Влияние

Google отличается от других компаний-разработчиков своим особым вниманием к *влиянию*. Инженер должен влиять на работу команды, а его работа должна влиять на продукт. От команды тестирования ожидают большого *влияния*. Коллективная работа команды должна быть не просто исключительной, а обязательно должна делать лучше и продукт, и саму команду.

Целью любого отдельного инженера и всей команды должно быть реальное *влияние*. Именно тест-менеджер отвечает за то, чтобы команда тестирования оказывала реальное *влияние* в компании.

Решения о повышении основываются на том, какое *влияние* специалист оказал на свой проект. Во время ежегодных отчетов менеджеров просят описать ценность вклада своих подчиненных и перечислить, на что это *повлияло*. Мы ожидаем, что при движении по карьерной лестнице сотрудник *влияет* на все большее количество вещей. Так как тест-менеджер направляет работу своих инженеров, то он отвечает за их рост и, значит, должен уметь измерять их *влияние*.

Управлять *влиянием* команд тестировщиков и разработчиков в тестировании — работа тест-менеджера.

Важно, что мы ставим перед командами тестирования задачу именно так — *влиять*. Мы специально не требуем от тест-менеджера и его ребят обеспечить высокое качество продукта. Мы не просим их отвечать за своевременный выпуск продукта. Мы даже не будем винить тестирование, если продукт провалится или не понравится пользователям. В Google нет ни одной команды, которая смогла бы взять на себя ответственность за все это. Команда должна быть ответственна за понимание целей и графика проекта и обеспечивать свою работу с позитивным *влиянием* на эти вещи — вот все, что мы просим. В Google самый приятный комплимент — услышать в свой адрес, что ты *влияешь* на ход вещей (или, для руководителя, — что его команда *повлияла*).

Итак, что является важным фактором в ходе ежегодного рецензирования работы и принятия решений о повышении? Правильно — старое доброе *влияние*. Оно оценивается для каждого сотрудника соответственно его должности и зоне ответственности. Для рядового инженера — в рамках его части проекта, для менеджера — в масштабе его команды и продукта. Чем выше взирается человек, тем большего *влияния* от него требуют, вплоть до влияния на весь Google (но об этом позднее).

Это работа тест-менеджера — построить команду, способную *влиять* на ситуацию так, чтобы каждый ее участник вносил свой вклад в зависимости от своей роли и навыков. Причем менеджеры Google не обязаны следить за каждой мелочью в процессе тестирования, не должны стоять за плечом у разработчика в момент создания тест-плана или просматривать каждую строчку кода. Их дело — чтобы все задачи решались правильными инженерами, серьезно нацеленными на результат, которые знают, как подойти к проблеме и что нужно использовать для ее решения. Тест-менеджер как будто расставляет всех важных игроков по полю, а дальше отходит в тренерскую зону — игра началась.

Давайте помечтаем о команде тестирования, в которой каждый инженер способен выполнять работу, приносящую ценность, и работает с максимальной пользой. О тщательно оптимизированном процессе тестирования, в котором каждая выполняемая единица работы направлена на достижение цели. О команде разработки, которая осознает объем работы по тестированию и участвует в ней. Задача тест-менеджера превратить все эти мечты в реальность.

Наконец, у тест-менеджера есть еще одна обязанность, связанная с взаимодействием между командами. Тест-менеджер, особенно опытный, не должен быть зашорен настолько, чтобы не видеть, что творится за пределами его проекта. Google — компания, в которой десятки продуктов разрабатываются и тестируются одновременно. У каждого из таких продуктов есть один или несколько тест-менеджеров, в зависимости от размера и сложности, и каждый из них придумывает свои способы, как увеличить *влияние* команды. Хороший менеджер отслеживает все передовые методы работы, распространяет информацию о них и применяет сам. Ведь проще всего доказать результативность инструмента, если эффективно использовать его в нескольких продуктах.

Наши команды тестирования подтверждают репутацию Google как инновационной компании. Множество тестовых приемов и инструментов, которые мы создали и которые используются за пределами Google, тому доказательство. Это стало возможным, потому что наши тестовые команды связаны общим духом инноваций. Тест-менеджеры обмениваются опытом не потому, что им так приказано свыше, и не потому, что ежемесячная встреча добавлена в календарь. Они собираются и общаются потому, что не хотят упустить свой шанс попробовать новые полезные изобретения другой команды. Кто-то хочет остаться последним тестировщиком, не пользующимся новейшим инструментом? Кто-то не хочет работать быстрее? У нас таких нет!

Наличие инноваций — один из критериев оценки влияния команды. Как бы ни круто было пользоваться своим изобретением в своем проекте, крутизна увеличивается, когда его осваивает соседняя команда, потом другая, третья, пока оно не станет частью технологии тестирования всей компании. Взаимодействие между командами должно быть построено на инновациях, иначе оно не выдержит проверки временем.

Интервью с Анкитом Мехтой, тест-менеджером Gmail

Анкит Мехта вырос в менеджера из инженера по тестированию, который в основном выполнял работу разработчика в тестировании. Начало своей карьеры в Google он провел, закопавшись в коде и автоматизации. Свое первое серьезное повышение до менеджера он получил в проекте Gmail.

Gmail не для слабаков: в этом продукте очень много динамических частей, за которыми нужно следить. С одной стороны, Gmail интегрируется с другими технологиями Google, включая Docs, Calendar и т. д., с другой — Gmail должен взаимодействовать с форматами других почтовых сервисов. Здесь требуется огромная работа с базами данных, так как Gmail работает в облаке, а его пользовательский интерфейс доступен через любой браузер. Не верите, что это сложно? Тогда вспомните, что Gmail имеет сотни миллионов пользователей, которые ожидают, что почта будет работать быстро, надежно, безопасно, а заодно справится с потоком спама. Добавляя новые фичи, надо сохранять рабочее состояние старых, что здорово затрудняет тестирование. Если в Gmail появляется баг, то весь мир узнает об этом мгновенно. Помидоры летят во многих в Google, но в первую очередь в находящегося на передовой менеджера.

Мы встретились с Анкитом, чтобы узнать, как было организовано тестирование Gmail.

— Расскажи, как ты подходишь к новому проекту по тестированию. Что ты делаешь в первую очередь, какие вопросы задаешь?

Анкит: Первые несколько недель в проекте я только слушаю. Очень важно ориентироваться, узнать архитектуру продукта и динамику команды. Я бы не стал слушать врача, который в первые пять минут моего визита выписывает мне антибиотики. Так же и команда не станет работать со мной, если я с ходу начну «ставить диагноз». Чтобы получить право выписывать лекарства, нужно многому научиться.

— Мы работали с тобой, и у нас сложилось впечатление, что ты не из молчунов. Наверное, когда ты начнешь говорить, тебя уже не остановить!

Анкит: Все так! Но у меня есть своя схема. Я понял, что самый сильный вопрос, который можно задать, «почему?». Когда я выхожу из роли молчуна, я обычно начинаю с него. Почему вы выполняете эти тесты? Почему вы написали этот тест? Почему вы автоматизируете эту задачу, а не другую? Почему мы тратим ресурсы на разработку этого инструмента?

Я думаю, что люди склонны что-то делать только потому, что видели, как это делают другие, или тестировать какую-то фичу потому, что точно знают — они с ней справятся. Если не спрашивать их «почему», они так и будут действовать по инерции, не объясняя себе свои действия.

— Какие же ответы на вопрос «почему» для тебя приемлемы?

Анкит: Всего два ответа. Во-первых, потому что это улучшает качество продукта. Во-вторых, потому что это повышает производительность инженеров, работающих на проекте. Все остальные ответы менее значимы для продукта.

— Команда Gmail известна своей целенаправленностью и производительностью. Теперь понятно, откуда берутся эти качества. Какие советы ты дашь менеджерам по тестированию для формирования здоровой рабочей среды?

Анкит: Очень важна динамика команды. Я считаю, что качество продукта связано с качеством команды тестирования. Сначала подберите нужных людей, с правильными навыками и настроем, а потом направьте их делать правильные вещи. Вы — руководитель команды, и именно ваше влияние создает здоровую рабочую культуру. Работая с Gmail, я формировал команду шесть месяцев, потому что мне хотелось создать сплоченную группу людей, где все понимают и уважают роли друг друга. Если вам удалось создать крепкую команду, она выдержит даже присутствие пары специалистов, которые не очень в нее вписываются.

Хорошие отношения между группами разработки и тестирования — важная часть положительной динамики всей команды. Когда я пришел в проект, ситуация была плачевной. Команды работали изолированно друг от друга, а полезные предложения тестировщиков терялись в команде разработки. Атмосфера была нездоровой.

— Судя по всему, тебе удалось исправить ситуацию, давай поговорим об этом. Расскажи, что ты сделал для того, чтобы рабочая среда стала комфортной?

Анкит: Когда я присоединился к проекту, команда тестирования была сосредоточена на тестах WebDriver, которые выполнялись для каждой сборки. Они проводили тесты, следили, как состояние меняется с зеленого (тесты проходят) на красный (тесты не проходят), и прикладывали титанические усилия к исправлению тестов, чтобы они не показывали ложные срабатывания. Команда разработки не сомневалась в том, что вся работа построена правильно, потому что тесты действительно обнаруживали важные проблемы, а значит, были оправданы. Однако иногда изменения затрагивали большие объемы кода, и оперативно исправить тесты не получалось. Чтобы достичь результатов, приходилось выполнять слишком большую работу. Весь процесс был ненадежным и недостаточно гибким для такого подвижного и сложного сервиса, как Gmail.

Так как я был новичком в проекте, мой взгляд не был замылен. Мне казалось, что самой серьезной проблемой Gmail были задержки. Серьезно, ведь для пользователя главный атрибут Gmail — скорость. Я решил, что если мы справимся с этой проблемой, мы сможем заслужить уважение разработчиков и они будут воспринимать нас как равных.

Задача была тяжелой. В каждой сборке мы сравнивали ее скорость со скоростью предыдущей версии, чтобы поймать случай, когда новая сборка работает медленнее. После мы перебирали все изменения кода в новой версии, чтобы найти и уничтожить причину задержки. Это был долгий и сложный путь с большим количеством проб и ошибок.

Я вместе с одним разработчиком в тестировании искал способ специально замедлить работу Gmail. Мы хотели лучше контролировать взаимодействие между клиентской частью и дата-центром, чтобы найти регрессии, влияющие на производительность. Собрав несколько старых компьютеров и лишив их всех ускоряющих компонентов, мы получили целую комнату системных блоков с 512 мегабайтами памяти, 40-гигабайтными дисками и медленными процессорами. Работа Gmail стала медленной, и мы смогли отличить сигнал от шума. Мы запустили долгосрочные тесты, которые создавали дополнительную нагрузку системе.

Первые месяцы были самые тяжелые. Пользы было мало, а вот ложноположительных срабатываний достаточно. Мы тратили много сил на настройку инфраструктуры и не получали результата. Но в один прекрасный день мы заметили деградацию системы. Мы смогли измерить задержки до миллисекунд и подкрепить слова данными. Разработчики теперь могли обнаруживать порождающие задержку регрессионные баги за часы, а не за недели, как раньше, и отлаживать их, пока они свежие.

Нам удалось заработать уважение для команды тестирования, и когда мы взялись за следующие высокоприоритетные задачи (исправление сквозных тестов

и налаживание инфраструктуры тестирования фактической нагрузки), разработчики сами вызывались помочь нам. Вся команда убедилась в пользе эффективных тестов, выпуски Gmail из ежеквартальных стали еженедельными, а потом и ежедневными.

— Отличный урок: возьмите трудную задачу, решите ее, заслужитеуважение. А что ты делал после этого?

Анкит: Трудную задачку можно найти всегда! Хотя общая идея остается в том, чтобы сосредоточиться на самом важном. Мы обозначили самую серьезную проблему Gmail, собирались вместе и решили ее. Затем мы пошли дальше по списку, и все решалось легко, так как у нас была сплоченная команда. Все-таки я больше верю в пользу сфокусированной работы. Когда я вижу, как команда пытается прыгнуть выше своей головы (скажем, работает над пятью задачами одновременно, хотя каждую выполняет только процентов на восемьдесят), я заставляю их сделать шаг назад и расставить приоритеты. Пусть количество задач будет меньше, но зато они будут решены на все сто. Плюс для команды: появляется чувство удовлетворения от выполненной работы, незавершенные задачи перестают нависать у них над головой. Плюсы для меня: качество продукта улучшается и я счастлив.

— Менеджеры в Google имеют много подчиненных, но и сами обязаны вносить технический вклад в проект. Как ты совмещаешь эти занятия? Можешь ли рассказать нам о своей инженерной работе?

Анкит: Обилие подчиненных и время, которое тратится на координацию работы команд, сильно отвлекает. Чтобы сохранить техническую квалификацию и оставаться инженером, я сделал две вещи.

Во-первых, я свел команды разработчиков и разработчиков в тестировании вместе. Работы там достаточно для всех, и я смог сташить кусочек для себя. Я занимался проектом с фазы проектирования и работал в нем достаточно, чтобы самостоятельно писать тесты.

Во-вторых, и это ключевая штука, если вы собираетесь заниматься технической работой, нужно абстрагироваться от всех отвлекающих факторов руководства. Сначала я просто выделял пару дней в неделю для самостоятельной работы. Взглянуть на тестирование глазами разработчика мне помогла работа по интеграции Google Feedback в Gmail. Каждый раз, когда я сталкивался с ненадежным тестом или неудобной частью тестовой инфраструктуры, я понимал, как наша работа выглядит для разработчиков. И все же, находясь в центральном офисе Google, полностью абстрагироваться от управлеченческой работы я не мог, поэтому я перебрался к команде Gmail в Цюрихе. Находясь в девяти часовых поясах от постоянного места работы, там, где я не являюсь начальником, я сумел влиться в инженерную команду и сделать очень много полезного!

- Можешь порекомендовать, как правильно комплектовать команду? Сколько должно быть разработчиков и тестировщиков? Как насчет соотношения разработчиков в тестировании и инженеров по тестированию?

Анkit: Главное правило найма — никаких компромиссов. Нанять неподходящего человека всегда хуже, чем подождать идеального кандидата. Нанимайте только лучших, и точка. Google не освещает соотношения инженеров в команде, но скажу, что раньше на нашем проекте тестировщиков было больше нормы. Решив первоочередные проблемы, мы увеличили долю участия разработчиков и вернулись к средним по Google показателям. Для Gmail хорошо сработал следующий вариант: 20% тестировщиков занимались исследовательским тестированием. Любой продукт, в котором взаимодействие с пользователем на первом месте, должен проходить исследовательское тестирование. Еще 30% составляли инженеры по тестированию, которые были сильнее сконцентрированы на продукте, они работали рука об руку с разработчиками в тестировании и обеспечивали максимальный результат их работы. Оставшиеся 50% составляли разработчики в тестировании, работающие над автоматизацией, добавлением инструментов для поддержания кодовой базы в рабочем состоянии и улучшением производительности разработчиков. Не знаю, буду ли применять такую разбивку ролей в следующем проекте Google, но для Gmail она сработала хорошо.

- Мы знаем, что тебе поручено тестирование Google+. Какие ценные уроки тестирования Gmail помогут тебе в работе с новым продуктом?

Анkit: Не стоит тратить все силы на проработку клиентской части. Gmail имеет, пожалуй, самую большую серверную часть, в которой было немало вкусных задач для тестирования. Кроме этого, были и другие полезные уроки:

- Пишите тесты на том же языке, на котором написано приложение.
- Человек, написавший функцию, должен быть ответственным за выполнение тестов для нее. Проигнорированные тесты выйдут вам боком.
- Создавайте инфраструктуру, в которой писать тесты будет легко. Тесты должно быть проще написать, чем пропустить.
- Примерно 20% всех сценариев использования отражают 80% самого использования. Автоматизируйте эти 20% и не беспокойтесь об остальных. Оставьте их для ручного тестирования.
- Мы все-таки в Google, а здесь важна скорость, ее ждут от нас пользователи. Убедитесь, что продукт работает быстро. Выделите это, если вам нужно доказать это остальным.
- Сотрудничество с разработчиками совершенно необходимо. Если его не будет, тестирование станет простым сглаживанием дефектов, а это не самая лучшая работа.

Иновации — часть ДНК Google. Команда тестирования должна иметь ген инноваций в крови, каждый участник должен изобретать новейшие решения проблем.

— Есть ловушки, в которые попадают инженерные команды?

Анкит: Да. Они думают, будто знают, чего хотят пользователи, и могут серьезно изменять продукт или новые фичи, не проводя экспериментов. Зачем вам хорошая тестовая инфраструктура для фичи, которую выкинут за ненадобностью? Дайте нескольким пользователям новую версию, получите обратную связь, а уж потом вкладывайтесь в грандиозную автоматизацию тестов.

Пока вы будете долго строить идеальное решение, рынок оставит вас позади. Работайте короткими итерациями, двигайтесь небольшими шагами, но двигайтесь!

Наконец, нужно найти идеальный момент для тестирования. Напишете тесты слишком рано — архитектура изменится, и вся работа коту под хвост. Затянете — вам придется переносить выпуск из-за нехватки качественного тестирования. TDD вам в помощь!

— А что же про конкретных людей? Ты заметил какие-нибудь типичные ошибки молодых инженеров по тестированию и разработчиков в тестировании?

Анкит: Ошибка новичков — отчаянно бросаться в работу. Они неистово пишут тесты, не обдумывая их назначение и место в общем процессе тестирования. Они не всегда понимают, что нужно брать ответственность за тесты, которые они приучили. То есть написали. *Ошибка разработчиков в тестировании в том, что они пытаются работать за разработчиков.* Разработчики в тестировании должны помнить, что тестировать код — задача разработчика, а их задача — ввести тестирование в рабочий процесс разработчика. Мы создаем условия, чтобы разработчик занимался сопровождением не только кода, но и тестов. Пусть разработчик в тестировании сосредоточится на ускорении выполнения тестов и предоставлении качественной диагностической информации.

Ошибка инженеров в тестировании в том, что они начинают вести себя как разработчики в тестировании. Мы же хотим, чтобы инженер по тестированию действовал глобально, контролировал весь продукт. Он должен смотреть на тесты с точки зрения пользователя, помогал обеспечивать эффективное использование всей тестовой инфраструктуры. Инструменты и средства диагностики, которые используют тестировщики, должны влиять на весь продукт.

— Какие еще успехи в области тестирования принесли пользу Gmail?

Анкит: Автоматизация JavaScript. Мы встроили сервлет автоматизации прямо в Gmail, и разработчики смогли писать сквозные тесты на том же языке, на котором была написана клиентская часть. Разработчики легко освоились с написанием тестов, потому что в них использовались те же методы и библиотеки — не при-

шлось тратить время на изучение. Стало просто писать тесты и определять, не нарушает ли новая возможность работу Gmail, и тем самым защищать старые фичи. Каждая функция Gmail теперь выпускается по меньшей мере с одним тестом, написанным с помощью этого сервлета. Кстати, сейчас я использую такую технологию в своей новой работе над Social. У нас уже более 20 тысяч автоматизированных тестов!

Еще одно достижение — нагружочное тестирование. В Google без него никак, потому что все наши приложения интенсивно используются, а дата-центры сильно загружены. По сути, мы должны смоделировать типичную нагрузку с нашим обычным трафиком. Мы потратили месяцы на анализ реального использования приложений, чтобы построить модель поведения пользователей. Затем мы сделали модель более реальной, проведя тестирование нагрузки на таких же компьютерах, на каких работают дата-центры. Параллельно были запущены две серии тестов: экспериментальная и контрольная. Мы наблюдали за ними, чтобы найти различия. Мы обнаружили много регрессий и передали их в руки разработчиков.

Наконец, наша ориентация на предотвращение багов (вместо их выявления) принесла плоды. Мы на ранней стадии включили автоматизированные тесты в процесс, предшествующий выпуску, и предотвратили попадание в сборку некачественного кода. Благодаря этому команда тестирования продолжает опережать ожидания и работает над сборками такого высокого качества, что нашим исследовательским тестировщикам есть над чем потрудиться.

— Итак, ты набил руку и теперь переходишь в социальные сервисы. На что ты будешь обращать внимание при найме участников в эту команду тестирования?

Анкит: Я хочу найти людей, которые не боятся сложностей. Тех, кто, сталкиваясь с большой проблемой, сможет разбить ее на конкретные шаги и, конечно, справится со всеми! Мне нужны люди, которые от жестких сроков не впадают в панику, а начинают действовать более эффективно. В моей команде должны быть люди, которые держат баланс между инновациями и качеством, чьи идеи выходят за рамки поиска багов. Кроме всего, я хочу видеть энтузиазм. Мне нужны люди, которые искренне хотят быть тестировщиками.

— Мы подходим к последнему вопросу. Почему ты выбрал тестирование?

Анкит: Мне нравится жонглировать быстрыми итерациями и высоким качеством — двумя на первый взгляд конфликтующими, но одинаково важными целями. Это классическое противостояние, которое заставляет меня оптимизировать оба направления — без ущерба для любого из них. Сделать новый продукт легко. А вот сделать его быстро и качественно — это целое испытание, которое стимулирует мою профессиональную жизнь и делает ее такой интересной.

Интервью с Хуном Даном, тест-менеджером Android

До того как Хун Дан был приглашен возглавить команду тестирования Android, он успел успешно поработать инженером в Apple и TiVo¹.

Стратегическую важность Android для Google трудно отрицать. Этот проект стал таким же хитом, как Search или Ads. Android получил особое внимание руководства и привлек первоклассных специалистов. Его кодовая база растет на глазах, и одна сборка может насчитывать сотни изменений в день. На базе этой операционной системы запускаются тысячи приложений, вокруг нее формируется сообщество разработчиков. Количество устройств, работающих на платформе Android, постоянно растет; она стоит на новых телефонах и планшетах множества производителей. В зоне ответственности Хуна и его команды все: от тестирования совместимости устройств и управления питанием до проверки работы приложений.

Мы встретились с Хуном, чтобы узнать, как организовано тестирование Android.

— Расскажи нам о том, как начиналось тестирование. Наверное, на первых порах, до появления смартфонов и многочисленных приложений, было проще?

Хун: К сожалению, нет. Когда я взял на себя руководство тестированием Android, команда только сформировалась и многие инженеры никогда не тестировали до этого операционные системы, не говоря уже о мобильных устройствах. Моей первой задачей было скоординировать работу команды и построить инфраструктуру тестирования. Я думаю, что начальная фаза проекта тестирования — самая трудная часть работы. Когда у тебя есть крепко сбитая команда и наложенная инфраструктура, то можно справиться с продуктом любой сложности. Тяжелее всего в планировании, зато потом легко в тестировании!

— Давай остановимся здесь, ведь множество менеджеров по тестированию бывают на ранних стадиях своих проектов прямо сейчас. Расскажи, что вы делали на ранней стадии проекта Android?

Хун: Серьезных задач было много. Самая первая — вникнуть в продукт. Я требую от тестировщиков досконального знания продукта. Первым делом нужно тщательно освоить то, с чем работаешь. Каждый участник моей команды должен знать структуру продукта, и точка. Когда ваши знания достигнут нужного уровня, самые сложные проблемы тестирования окажутся на поверхности и вам, оттал-

¹ TiVo — компания, которая разработала цифровое устройство для записи видео — TiVo. Устройство по сети связывается с сервером и записывает выбранные пользователем телепередачи или сериалы, которые он может просмотреть в любое время на своем телевизоре.

киваясь от них, будет легко формировать команду. Нанимайте или привлекайте людей, которые справляются с самыми сложными задачами проекта. Структура Android очень разветвленная: от аппаратного уровня до ОС, фреймворка, модели приложений и магазина. В этом проекте много динамических частей, требующих особой квалификации тестирования. Поэтому первое, что я сделал, — это выявил их и собрал такую команду, которая бы с ними справилась.

Когда команда была сформирована, я отдал приказ: «Приносите пользу!» Точнее, приносите пользу постоянно. Все, от разработчика до руководителя, должны содействовать тестированию, находиться в общем потоке. Делаете немного меньше — и вы уже мешаете. На раннем этапе общая задача — способствовать успешному запуску продукта. Я не буду спорить: многое из того, что мы делали, было за рамками компетенции тестировщиков, но вся наша работа шла проекту на пользу, и мы выпустили качественный продукт. Мне удалось расставить правильных людей по своим местам, и мы синхронизировали рабочий процесс. Стала возможна ежедневная сборка, конфликты были исключены, и даже сообщения о багах моя команда отправляла в едином стиле. Все дело в правильно подобранный команде.

— Звучит впечатляюще. И как командаправлялась?

Хун: На самом деле я потерял около 80% старой команды. Зато оставшиеся стали техническими лидерами. Они подавали пример и работали с новичками, которых мы привлекали в команду. Нет универсальных людей, которые вписываются во все проекты. В Google достаточно работы, и если кому-то не подошел Android, то, возможно, подойдет другой проект. В Google такое разнообразие и создает здоровую среду тестирования.

— Какие были приятные моменты тестирования?

Хун: Приятно, что нам удалось сфокусироваться на пользе. Все, что мы делали, имело цель. Мы ставили под сомнение все: каждый тестовый пример, каждый блок автоматизации. И многое из того, что было сделано раньше, не проходило этой проверки. Любую часть автоматизации, у которой не было ясной цели, мы тут же выбрасывали. Все крутилось вокруг пользы. Если бы мне предложили дать совет начинающим менеджерам по тестированию, я бы сказал: все, что вы делаете, должно приносить максимальную пользу и приносить ее неоднократно.

— Это непросто, хотя по твоим словам этого и не скажешь! Расскажи нам более подробно об организации тестирования, о которой ты обмолвился. Ты коснулся процесса сообщения о багах, но, конечно, это не все. Как ты организуешь саму работу?

Хун: Я воспринимаю структуру Android как своеобразную «колоннаду». Я знаю, что при тестировании Chrome OS вы используете другую терминологию, но мне нравится идея тестовых колонн. Каждая группа тестировщиков отождествляет-

ся с одной из таких колонн, на которых держится весь продукт. Для Android мы организовали четыре колонны: системную (ядро, носители информации и т. д.), колонну фреймворков, колонну приложений и колонну магазина приложений. Спросите любого из моих тестировщиков, что они тестируют, и вам укажут одну из этих колонн.

— Логично, что колонны соответствуют разным навыкам тестировщиков. Одни хорошо работают с низкоуровневыми аспектами, другие лучше справляются с высокоуровневыми, а вместе они поддерживают всю систему. Хорошая схема. А как насчет автоматизации? Как она вписывается в вашу задачу?

Хун: Я научился скептически относиться к автоматизации. Тестировщики могут загореться идеей автоматизации продукта и потратить месяцы на ее создание, а потом продукт или платформа меняется, и все, что они сделали, идет прахом. Нет более бессмысленной траты ресурсов. На мой взгляд, автотесты должны быстро создаваться, быстро выполняться и решать узкую задачу. Если цель автоматизированного теста непонятна сразу, значит он слишком сложен. Сделайте автоматизацию простой, ограничьте ее масштаб, а самое главное — следите за тем, чтобы она приносila пользу. Например, у нас есть пакет автоматизированных тестов, который проверяет сборку на возможность перехода из канареичного канала в Droid-канал.

— Минуточку! Что за Droid-канал?

Хун: О, извини. Как я уже упоминал в начале разговора, в проекте Android некоторые вещи делаются иначе. И называются тоже по-другому — мы используем не те каналы, к которым вы привыкли в Chrome. У нас есть канареичный канал, Droid-канал (аналог канала внутренних пользователей, но для команды Android), экспериментальный канал, Google-канал и уличный канал (сборка, которую мы передаем для внешнего использования). Идея та же, просто другое название.

— Итак, у вас есть пакет автоматизированных тестов для выявления багов, препятствующих переходу канареичной сборки в канал для разработчиков. Этими тестами занимаются выделенные разработчики в тестировании?

Хун: Ни в коем случае. У меня нет ни одного специализированного тестировщика. Все занимаются ручным тестированием, буквально все. Исследовательское тестирование — лучший способ хорошо изучить продукт. Я не хочу, чтобы мои разработчики в тестировании только и делали, что писали фреймворки. Я хочу, чтобы они были вовлечены в продукт и знали, как его использовать. Каждый тестировщик должен смотреть на продукт глазами пользователя, должен быть экспертом, разбирающимся во всех тонкостях продукта. Автоматизации мы оставляем тестирование стабильности, управления питанием, производительности, нагрузки,

ну и еще быстрые проверки. Не нужен человек, чтобы обнаружить утечку памяти из-за использования камеры или чтобы проверить одну и ту же фичу на всех платформах. Действия, которые требуют многократного повторения или машинной точности, недоступной для людей, — вот где нужна автоматизация.

— Так значит, у вас больше тестировщиков, чем разработчиков в тестировании?

Хун: Нет, наоборот, разработчиков в тестировании почти вдвое больше. В моей команде каждый разработчик в тестировании может играть роль инженера по тестированию, когда это нужно, и наоборот. Я не обращаю особого внимания на должность человека, главное, чтобы он приносил пользу.

— Что ж, поговорим о ручном тестировании, потому что ты явно относишься к нему серьезно (и это восхищает!).

Хун: Я верю в пользу целенаправленного ручного тестирования. Действовать наугад неэффективно. Мы внимательно присматриваемся к ежедневной сборке, которую тестируем, и анализируем ее содержимое. Что изменилось? Сколько изменений добавилось или трансформировалось? Какая функциональность добавилась, а какая изменилась? Кто из разработчиков отправлял изменения? Насколько обширны изменения по сравнению со вчерашней сборкой? Это помогает сфокусироваться, и вместо рысканья по всему продукту мы сосредоточены на проверке изменений изо дня в день, что и делает нашу работу намного более производительной.

Понятно, что координация в команде очень важна. Так как я настаиваю, что каждый участник должен заниматься исследовательским тестированием, нужно свести к минимуму перекрытия. На планерках мы уточняем, что нужно тестировать, и определяем, кто чем занимается. Я считаю ручное тестирование успешным, если оно целенаправленно и скоординировано. Если эти условия соблюdenы, значит время на тестирование потрачено с пользой.

— Вы создаете документацию для ручного тестирования или занимаетесь исследовательским тестированием?

Хун: Мы занимаемся исследовательским тестированием и создаем документацию. Ручные тест-кейсы документируются в двух случаях. Первый — если один и тот же сценарий использования повторяется в каждой сборке или входит во все серии тестов. Мы записываем их и заносим в базу тестов GTSM, чтобы любой внутренний или внешний тестировщик мог их использовать. Второй случай — документирование рекомендаций по тестированию для отдельных фич. Каждая фича обладает собственными уникальными свойствами. Тестировщики записывают их как рекомендации для коллег, которые могут работать с этой фичей в следующей сборке. Итак, у нас есть документация двух видов: описание общих сценариев и рекомендации по конкретным фичам.

— Расскажи нам о своих требованиях к разработчикам. Ты требуешь от них спецификаций? Заставляешь применять TDD? Как насчет юнит-тестов?

Хун: Наверное, где-то есть сказочный мир, в котором каждой строке кода предшествует тест, а ему — спецификация. Не знаю, может, такой мир и существует. Но в стремительном современном мире, в котором я живу, приходится работать с тем, что имеешь. Есть спецификация? Отлично! Большое спасибо, пригодится! Давайте будем реалистами: нам нужно искать пути, как работать в существующей системе.

Если вы требуете спецификацию, это не значит, что вы ее получите. Если вы настаиваете на юнит-тестах, это еще не значит, что они принесут пользу. Ни написание спецификации, ни создание юнит-теста не поможет вам найти проблему, с которой столкнется реальный пользователь (кроме обнаружения очевидного регрессионного бага). Это законы моего мира — мира тестировщиков. Вы работаете с тем, что дано, и выжимаете из этого максимальную пользу для продукта и команды.

По моему опыту, у всех инженеров благие намерения. Никто не хочет создавать продукт с багами. Но инновации нельзя точно спланировать. Графики и конкурентное давление существуют независимо от моих жалоб на качество. Я могу жаловаться, а могу заниматься полезным делом. Предпочитаю второй вариант.

Я живу в мире, где каждый день выходят сотни изменений. Да, я сказал *каждый день*. Талантливым, прогрессивным разработчикам нужно столько же талантливых, прогрессивных тестировщиков. Для требований и капризов нет времени — только польза.

— Хун, мы рады, что ты на нашей стороне! А теперь блиц-опрос. Если бы тебе дали лишний день для тестирования выпуска Android, что бы ты сделал?

Хун: Если бы у меня был лишний день, я бы сделал еще одну сборку! Нет такого понятия — лишний день!

— Туже! Тебе есть о чем жалеть? Есть баг, который пробрался через канал выпуска к пользователю?

Хун: Прежде всего, такое может произойти с каждым тестировщиком на планете. Ни одна компания не выпускает продукт, полностью лишенный багов. Такова природа вещей. Но для меня все такие случаи были болезненными.

— Нет, ты так легко не отвертишься! Назови хотя бы один!

Хун: Ладно, был у нас баг с активными обоями (Active Wallpaper) несколько выпусков назад. Иногда при запуске обоев все просто падало. Решение было простым, а мы выдали его так быстро, что пользователь почти ничего не заметил. Но это нас не оправдывает. Мы написали тесты для этого случая, поверьте мне, мы их написали!

Интервью с Джоэлом Хиноски, тест-менеджером Chrome

Джоэл работал тест-менеджером в Google с первых дней открытия офиса в Сиэтле (Кирклэнде) и успел поруководить множеством команд за эти годы. Сейчас он отвечает за все клиентские продукты, включая Chrome и Chrome OS. Джоэла прозвали «капризным австралийцем», и, возможно, это характеризует его требовательный стиль тестирования, но совсем не вяжется с его высокими рейтингами как руководителя.

Недавно мы встретились с Джоэлом и обсудили его представления о тестировании и опыт работы с Chrome и Chrome OS.

— Признавайся, на каком компьютере ты работаешь!

Джоэл (достает ноутбук): Chromebook — вот моя детка!

— Сколько еще гаджетов затерялось в твоем рюкзаке?

Джоэл: Ха! У меня в кармане сотовый телефон, а где-то рядом лежал планшет. Я использую свой Chromebook для работы, а когда он делает что-то не так, я сообщаю об ошибке. Надо пользоваться оборудованием, которое тыatestишь!

— Итак, ты руководишь тестированием и управляешь полным спектром продуктов: панели инструментов, инсталлеры Chrome и Chrome OS и все, что работает на клиентской операционной системе (и сама операционная система), — твои зоны ответственности. Тебе приходится управляться с множеством команд разработчиков. Как ты сохраняешь баланс?

Джоэл: Само тестирование — искусство поиска баланса. Одной рукой мы готовим продукт к выпуску, тестируем каждую его версию и проводим необходимые проверки. Другой рукой мы создаем процесс автоматизации, ее инфраструктуру и выделяем ресурсы на разработку фреймворков. Дополнительная рука нам понадобится, чтобы запланировать и сформировать четкую структуру процесса тестирования от разработки до выпуска. Ну и когда гуру тестирования рассказывают о своих революционных методах работы, нужно откуда-то взять еще одну руку для экспериментов.

— Ты жонглер или осьминог? А если серьезно, какое место ты занимаешь в этой многоходовой комбинации?

Джоэл: Стараюсь быть практическим. Мне нужно выпустить программу, а для этого придется чем-то пожертвовать. Это всегда торги. Конечно, у нас гибкая система разработки, но все равно приходится делать обязательную финальную проверку качества (мы называем ее «последняя миля»). Мы проводим исследо-

вательское тестирование, но все равно отслеживаем разные версии и платформы. Все относительно.

Не бывает универсальных моделей тестирования, даже внутри одной компании условия различаются. Пример: мои команды Chrome и Chrome OS используют разные процессы, хотя работают в одном здании! Может, одна команда лучше другой? Не думаю. Я помогаю командам обмениваться информацией о рабочих методах, что делает нашу работу по тестированию более производительной. Моя команда тестирования должна быть готова ко всему, знать, какие приемы работают, а если она сталкивается с неработающими методами, то должна легко от них отказываться. Пока я не разобрался со всем этим до конца, я предпочитаю использовать гибридный метод — сочетание тестирования разработчиками, сценарного тестирования, исследовательского тестирования, тестирования на базе оценки рисков и функциональной автоматизации.

— Кажется, на горизонте маячит еще одна книга по организации тестирования в Google.

Джоэл: Да, дайте мне год, и мы сравним объемы продаж или рейтинги на Amazon. Хотя какого черта, мы в Google — мы померяемся релевантностью!

— Ладно, вернемся к тестированию в Chrome и Chrome OS. В этой книге мы обсуждаем общую инфраструктуру тестирования Google для команд веб-приложений. Ты ведь работаешь в клиентском пространстве, твоя работа сильно отличается?

Джоэл: Отличается, и это создает много проблем. Клиентское направление не основное направление в Google. Мы — веб-компания и знаем, как хорошо построить и протестировать веб-приложения. Основная проблема с клиентскими продуктами в том, чтобы преобразовать накопленный опыт и инструментарий для клиентских машин. Это серьезная трудность, ведь инфраструктура Google не может помочь моим проектам.

Chrome начался с небольшого эксперимента. Несколько разработчиков собрались вместе и решили построить качественный браузер, чтобы любой мог его использовать и изменять. На ранней стадии его тестировали разработчики и несколько хардкорных тестировщиков, которые стали первыми пользователями продукта. Но когда у вас десятки миллионов пользователей, вам бы лучше иметь чертовски хорошую команду тестирования.

— Мы узнаем ребят, которых ты описал. А после того как команда была сформирована, какая проблема стала самой серьезной?

Джоэл: Сам веб! Серьезно, среда постоянно меняется, а Chrome обязан поспевать. Поток дополнений, расширений, приложений, версий HTML и Flash не прекращается. Количество переменных факторов зашкаливает, а ведь все они должны

работать. Если мы выпустим браузер, который не отображает ваш любимый сайт или приложение, вам не придется долго искать альтернативный браузер.

Еще мы тестируем на разных операционных системах, но их не так много, а наша инфраструктура виртуализации упрощает процесс. Так что все проблемы, которые меня беспокоят, идут из веба.

— Да, разнообразие не на руку тестировщику. Мы понимаем, что ты будешь писать свою книгу, и не станем есть твой хлеб. Однако назови хотя бы две технологии или два решения, которые помогли тебе приручить веб.

Джоэл: Две? Хм... Ладно, я назову совместимость приложений и автоматизацию пользовательского интерфейса, потому что в этих областях мы добились больших успехов. Все остальное я приберегу для следующей книги, той, более успешной!

Совместимость приложений очень важна для браузеров. Совместим ли Chrome с приложениями и сайтами в вебе? Правильно ли в Chrome отображаются страницы и запускаются веб-приложения? Разумеется, мы не можем проверить все приложения и сайты. И даже если бы могли, с чем их сравнивать? Мы отвечаем на эти вопросы, тестируя самые популярные сайты (в Google эту информацию не нужно долго искать) в контрольных версиях Chrome и браузерах конкурентов. Наши инструменты автоматически открывают тысячи сайтов и сверяют правильность их отображения. Мы делаем это для каждой ежедневной сборки, поэтому регрессии обнаруживаются очень быстро. Если сайт отображается иначе, сразу же подключается человек, чтобы найти, в чем проблема.

Впрочем, это только вершина айсберга. Нам еще нужно уметь запускать сайты и приложения, а для этого нужны были средства автоматизации на уровне пользовательского интерфейса. Chrome поддерживает API, называемый Automation Proxy, который можно использовать для запуска браузера, открытия URL-адреса, проверки его состояния, получения информации о вкладках и т. д. Мы разработали для него интерфейс на Python, так что теперь можно написать скрипт для браузера на Python (этот язык знают большинство тестировщиков в Google). Так автоматизация функциональных тестов стала возможна, и мы создали большую библиотеку PyAuto¹, над которой работали и разработчики, и тестировщики.

— Ладно, ты победил Chrome. Так как Chrome OS — это тот же Chrome, только встроенный в ноутбук, его тестирование, наверное, было простым?

Джоэл: Можно подумать, если я тестирую Safari, то заодно тестирую и Mac OS? А если я протестировал IE, то и Windows летает? Ага, сейчас! Само существование Chrome OS только усложняет тестирование Chrome, потому что мне приходится добавлять еще одну платформу в список совместимости приложений.

¹ Почитать подробнее о PyAuto можно по адресу: <http://www.chromium.org/developers/testing/pyauto>

И все же я скажу вам, что у нас все схвачено, и это очень круто. Google контролирует все в этой системе с самого низа, от компонентов, работающих с материнской платой, и поднимаясь наверх до пользовательских интерфейсов. Поэтому спускаясь обратно вниз, видишь много знакомого — многое перекликается с тестированием Chrome. Я могу использовать PyAuto и строю хорошие автоматизированные пакеты с возможностями повторного использования результатов команды Chrome. А потом начинается: прошивка, ядро, графический процессор, сетевой адаптер, беспроводной модем, 3G... Чего только нет в этих миниатюрных устройствах! Здесь уже сложнее применять автоматизацию. Это трудоемкие задачи тестирования, которые не работают при стандартном для Google соотношении количества разработчиков к количеству тестировщиков. Здесь нужно больше тестирования. Мы работали с этими системами со стадии прототипа, и нам даже приходилось монтировать электронные платы на картонных коробках.

Известные инструменты тестирования нашей системе не подходили. Chrome OS работает с открытым кодом и находится за рамками обычной системы разработки Google. Нам пришлось заново изобретать колесо (причем почти буквально) во многих тестовых инструментах, менять процессы с учетом особенностей новых инструментов и предоставлять эти инструменты внешним разработчикам. Мы выпускаем эту ОС для пяти разных платформ на трех каналах (разработчиков, бета, стабильном) с шестинедельным графиком. Если бы я не был австралийцем, я бы сошел с ума!

Итак, нам пришлось творчески решать задачи. Где мы можем переложить создание инструментов на разработчиков? Какой объем тестирования можно требовать от партнеров и производителей? Как обучить нашу команду эффективно тестировать оборудование? Что можно придумать, чтобы снизить нагрузку на ручное тестирование? И как это будет соотноситься с запуском на реальных устройствах?

— То, что ты начал отвечать вопросами, говорит о том, что нам придется ждать выхода твоей книги для получения ответов!

Джоэл: Да, придется ждать, потому что даже у меня нет всех ответов, и это моя работа — найти их. Мы в самом начале пути, нам еще нужно найти подход, гармонично сочетающий ручное тестирование с автоматизацией. Мы переработали Autotest¹ (опенсорс-инструмент для тестирования ядра Linux) для управления большим набором автоматизированных тестов оборудования Chrome OS. Команда, работающая над этим, значительно расширила программу с учетом особенностей нашей платформы и внесла много изменений, причем все это в условиях открытого кода. В итоге Autotest выполняет наш цикл предстартовых тестов, смоук-пакет, а также тесты проверки сборки на реальном оборудовании и на виртуальных машинах. Конечно, мы не забываем PyAuto и с его помощью управляем автоматизацией тестирования браузера Chrome, работающего в Chrome OS.

¹ Про Autotest можно почитать по адресу <http://autotest.kernel.org/>

— В Google хорошо известно: хочешь найти хорошего тестировщика — обращайся к Джоэлу или Джеймсу. Ты учили наших рекрутеров искать и нанимать сотрудников, а кандидатов, которые не были уверены, хотят ли они стать тестировщиками, отправляли к тебе для завершающей беседы. В чем тут магия?

Джоэл: Мы с Джеймсом давно работаем в одном офисе, и мы оба ревностно относимся к тестированию. Однажды мы объединили свои силы. Джеймс — очень общительный, часто выступает на конференциях, кажется, что он знаком со всеми. Круг его общения велик и работает на него. А на меня работает то, что я умею заинтриговать в людях интересом к тестированию. Это совершенно другой подход, как разница между везением и мастерством!

Конечно, это шутка, но я действительно ревностно отношусь к тестированию и поэтому готов нанимать только правильных кандидатов на роли инженеров Google. Например, найти людей для Chrome было непросто из-за тенденции бросать все силы на решение проблемы. Возникли неполадки с проверкой CR-48, Samsung Chromebook и новых экспериментальных платформ по трем каналам за неделю? Бросаем тридцать специалистов, пускай разбираются! Нужно проверить стабильную сборку Chrome S за двадцать четыре часа? С восемнадцатью ручными тестировщиками это будет легче легкого!

Я не хочу управлять командой тестировщиков, которые слепо следуют тестовым сценариям. Это скучно. Я хочу управлять командой, которая занимается революционными разработками, создает новые инструменты, подходит творчески к решению рутинных задач. Добавляя участников в команду, я хочу сохранить ее высокий технический уровень. В этом состоит основная проблема с наймом. Нужно найти профессионалов, подходящих Google по техническим навыкам, и помочь им загореться тестированием. Конечно, Джеймс знает много таких людей, но когда-нибудь его сети опустеют. Поэтому я подхожу более основательно и стараюсь объяснить, что же делает тестирование такой интересной и творческой работой. Вы удивитесь, как много ребят, мечтавших стать разработчиками, захотели попробовать себя в тестировании. Когда они понимают, какая сложная и интересная эта работа, у меня появляется хороший тестировщик.

— И последний вопрос: зачем выбирать карьеру в области тестирования?

Джоэл: Тестирование — это последний рубеж проекта. Как сделать разработку эффективной — уже более-менее понятно, но есть непочатый край работы по решению проблем тестирования: от организации всей технической работы до создания быстрой и эффективной автоматизации. И все это с нужной гибкостью. Сегодня это самая интересная область разработки с потрясающими возможностями карьерного роста. Вам уже не придется возиться с кусочком кода, вы летаете выше: провели аппаратное ускорение своего сайта на HTML5, убеждаетесь в том, что ваша оптимизация ядер процессора обеспечивает максимальную производительность

и что ваша песочница¹ действительно безопасна. Меня такие вещи интересуют и воодушевляют, поэтому я доволен тем, что работаю над организацией тестирования в Google, более того, над одним из самых сложных проектов, которые у нас есть.

Директор по тестированию

Директор по тестированию в Google — личность самодостаточная. Трудно написать о нем целую главу и точно определить, кто же такой директор по тестированию, потому что каждый обладает полной независимостью и использует ее на всю катушку. У них есть лишь несколько общих черт. Например, они все подчиняются Патрику Коупленду. Все они работают в инфраструктуре Google, и раз в неделю директора по тестированию встречаются для обсуждения своих направлений. Однако в отличие от тест-менеджеров из предыдущего раздела (которыми они руководят), у директоров есть карт-бланш на руководство своими продуктивными командами.

Еще одна общая черта — все директора утверждают решения о найме и переводе сотрудников и контролируют все вопросы комплектации тестовых команд. Им выделяется бюджет на проведение тимбилдингов, выездов и покупку сувенирки с символикой Google: рюкзаков, футболок, курток и т. д. Тестировщики даже соревнуются, кто закажет самое крутое снаряжение для своего войска. Уже вошло в обычай заказывать больше, чем нужно, чтобы потом делиться с другими. У тестирования в Google и так серьезная репутация, а уникальная одежда помогает ее поддерживать. Иногда они даже вводят моду на какую-то вещь. Команда Уиттакера сделала футболки с надписью «The Web Works (you're welcome)²»; эти футболки стали настолько популярными, что их носили даже разработчики.

Это не попытка принудить команды к тотальной синхронизации и свести к минимуму работу, которая повторяется в разных командах. Мы ждем инноваций от всех команд, а соревновательность при разработке инструментов только делает команды сильнее. Тем не менее существуют регулярные и единовременные премии, которые стимулируют сотрудничество, поэтому свое «двадцатипроцентное» время инженеры часто тратят на другой проект, работая под началом другого директора. Часто директора используют «двадцатипроцентное время» для аккуратного перевода тестировщика в другую команду: сначала он проводит в новой команде 20% времени, а потом столько же тратит на работу в старой.

Если нужно выделить только одну вещь в Google, которая позволяет сохранить дух сотрудничества, несмотря на естественную склонность людей к соперничеству,

1 Песочница (от англ. *sandbox*) — отдельное пространство для тестирования программ или приложений. Любые игры в ней не повредят ОС, так как песочница изолирована от системы.

2 The Web Works (you're welcome) — с английского можно перевести как «Интернет работает (не стоит благодарности)». — Примеч. перев.

то это именно такой открытый процесс перехода. Мы поощряем переход инженеров в другую команду каждые полтора года. Именно поощряем, а не требуем! Директора поддерживают между собой хорошие отношения, потому что резерв рабочий силы у нас общий и циркуляция работников на руку всем.

Задача директора — быть лидером. Он должен собирать сильные команды и мотивировать их на выпуск качественных и полезных продуктов, способных изменить что-то в отрасли. Директор должен быть технически грамотным, чтобы пользоваться уважением инженеров, динамичным, чтобы не отставать от строительного стиля работы Google, и должен быть хорошим управленцем, чтобы поддерживать продуктивность команд. Директор должен быть ниндзя от инструментария и инфраструктуры Google, чтобы молниеносно принимать решения, необходимые для ежедневных выпусков.

Что помогает решать все эти задачи и быть успешным директором по тестированию в Google? Мы решили, что лучше всего нам ответят те люди, которые уже этим занимаются!

Интервью с Шелтоном Маром, директором по тестированию проектов Search и Geo

Шелтон Mar — директор по тестированию; эта должность является аналогом вице-президента в других компаниях. Он один из самых давних тестировщиков Google, человек, который пришел раньше Патрика Коуплена, в те времена, когда направление продуктивности разработки еще называлось просто службой тестирования. Шелтон вырос из тест-менеджера маленьких групп в директора, отвечающего за весь поиск, карты и инфраструктуру. Сейчас он руководит тестированием направления продуктов, которое Google называет Local and Commerce: все продукты, относящиеся к геолокации, включая Google Earth и Maps.

Мы встретились с Шелтоном, чтобы узнать о прошлом Google и о том, что было сделано для тестирования Google Search.

— Шелтон, ты долго работаешь в Google и, наверное, помнишь службу тестирования, о которой Патрик упоминает в предисловии. Расскажи, как выглядело тестирование в далекий докоупленский период?

Шелтон: Тогда, конечно, все было иначе, но одна вещь так и не изменилась — компания Google всегда могла работать в очень быстром темпе. В те дни нам везло, так как интернет был проще, приложения меньше, а группы умных людей,

просто работающих в полную силу, было достаточно. Было много авралов, но нескольких героев хватало, чтобы их преодолеть. В продуктах образовывались зависимости системного уровня, а сквозное тестирование проводилось и вручную, и автоматически. Чем больше мы росли, тем больше проблем вырастало из этих зависимостей.

Не хочу сказать ничего плохого о таком тестировании, оно является необходимой частью проверки правильности работы интегрированной системы, но чрезмерные зависимости на поздних циклах тестирования сильно усложняют отладку.

Мы пытались решить эту проблему, когда появился Пат.

— В бэкенд-системах, где труднее определить сквозной сценарий, ситуация была еще хуже?

Шелтон: Именно! Мы часто не могли выпускать бэкенд-системы так быстро, как хотели, потому что у нас возникали трудности с проверкой качества. От бэкенд-систем зависит много вертикалей продуктов, поэтому они должны работать правильно. Например, из-за ошибки в BigTable пострадает множество приложений. Обновление такой системы создает «эффект домино» из-за проблем, которые невозможно обнаружить только сквозными тестами.

— То есть ты заменил затратные сквозные проверки усиленными проверками серверной инфраструктуры бэкендов? Расскажи о своих шагах.

Шелтон: Сначала мы изменили состав команды. Мы переопределели роль разработчика в тестировании и сосредоточились на найме технически сильных специалистов. Когда нужные люди были собраны, мы взялись за создание лучшего решения для бэкенд-тестирования. Мы сосредоточились на автоматизации уровня компонентов. Представьте команду изобретательных инженеров с навыками разработки и тестирования, дружно создающих бэкенд-инфраструктуру, и вы поймете, как шли дела.

— Какой фактор был ключевым для вашего успеха?

Шелтон: Для нас было важно получить поддержку разработчиков. Наши разработчики в тестировании очень тесно работали со своими партнерами из разработки (я специально использую слово «партнер», потому что это была совместная работа, а вовсе не заслуга только тестирования). Ребята вместе создавали новые методы тестирования на уровне разработки, и вся наша работа становилась лучше благодаря такому партнерству. Когда какие-то задачи не решались на уровне компонентов, мы решали их уровнем ниже. Динамика группы изменилась, и теперь вся команда проекта (и разработка, и тестирование) отвечала за качество на уровне компонентов, а инженеры по тестированию сосредоточились на процессе, фреймворках, инструментах и интеграции.

— Ты принимал довольно рискованные решения — например, склонить высокопрофессиональных разработчиков к тестированию. Что это дало? Не сожалеешь ли ты об этом? Как это повлияло на культуру тестирования?

Шелтон: Вероятно, это стало самым важным нашим решением для Google. Мы осознали, какие вещи мы должны изменить в Google как можно раньше:

- преобразовать тестирование так, чтобы вся команда (и разработка, и тестирование) отвечала за качество продукта;
- тестирование должно стать неотъемлемой частью проектной команды. Значит, нам нужны сильные инженеры, отлично ориентирующиеся в технологиях;
- тестирование должно использовать современные компьютерные технологии и методы.

Чтобы добиться этого, нам нужны были умные и сильные инженеры-программисты, которые понимают в тестировании (или хотя бы могут научиться). Как только мы взглянули на задачу с этой стороны, мы поняли, что можем привлечь в нашу команду лучших инженеров, чтобы справиться с нашими головоломками тестирования. Тот факт, что мы собрали большую команду таких ребят, свидетельствует, что им действительно интересно.

— Во время своей работы в Google ты работал над многими проектами, включая ключевой для компании проект поисковой системы. Что было самым трудным в его тестировании?

Шелтон: Решить, на чем сосредоточить тестирование! Когда инженеры начали смотреть на тестирование Search, они стали говорить о том, что Google выдает в результатах поиска. Конечно, эта тема достойна исследования, но качество поиска подразумевает намного больше. Чтобы пользователь получал однозначные, надежные и быстрые ответы, мы должны проверить всю сложную распределенную программную систему, которая возвращает результаты. Тестировщик должен понимать алгоритмы индексирования и поиска. Тестировщик должен понимать всю подноготную системы, как она устроена, чтобы уметь проверять ее действия там, где они происходят, тогда, когда они происходят.

Мы сфокусировались на этом с самого начала. Мы разделили качество поиска и качество работы самой системы. Мы занялись последним, а качество поиска оставили экспертам из команды разработки. Разработчики следили за тем, чтобы Google возвращал лучшие возможные результаты, а мы следили за самой инфраструктурой поиска и организацией его выдачи.

— С чего ты обычно начинаешь работу над новым проектом? Что делаешь в первую очередь? С точки зрения формирования команды? С точки зрения технической инфраструктуры? С точки зрения процесса тестирования?

Шелтон: Обычно мой первый вопрос к команде: «Что самое важное для тестируемой системы?». Для поиска важно быстродействие, для новостей — актуальность, для карт — покрытие. Каждое приложение стоит на своих китах. Тип системы помогает нам определить ее важные аспекты: целостность данных важна для хранилища, возможность масштабирования — для сетевых программ, возможность совместного использования — для систем управления проектами.

Определили важные части тестируемого продукта? Направляйте на их проработку основные силы. Несложными компонентами (типа доработки пользовательского интерфейса и прочей мишуры) можно заняться после. Сосредоточьтесь на базовых атрибутах продукта, которые трудно изменить, и не тратьте много времени на вещи, которые изменяются легко. Например, если тестировщик рано начал заводить баги, пойманные в шрифтах, я решу, что он неправильно расставил приоритеты.

— Между ручным и автоматизированным тестированием всегда существовали трения. Похоже, Google от высокой доли ручного тестирования переходит к высокой доле автоматизированного тестирования. Что ты думаешь по этому поводу? Какая пропорция правильна? Как ты понимаешь, что перестарался в том или ином направлении?

Шелтон: Я считаю, следует автоматизировать как можно больше. Мы используем концепцию непрерывной сборки, при которой ручное тестирование только мешает. Проверка на уровне компонентов и интеграционные тесты делают то, с чем ручное тестирование не справится. С другой стороны, автоматизация недостаточно гибка к изменениям и требует сопровождения. Технологии стремительно изменяются, и автоматизацию тоже нужно дорабатывать, чтобы успевать за ними.

Должен сказать, что ручное тестирование тоже в чем-то уникально. В области мобильных приложений, например. Изобилие оборудования, экранов, форм-факторов и драйверов порождает разнообразие в отображении и визуализации. Нам приходится тестировать эту область вручную. Но мы все равно стараемся автоматизировать и этот процесс, насколько это возможно. Пусть машина выполнит 90% работы, а человеческие ресурсы мы подключим только на последних 10% цикла проверки, которые мы называем «последней милей». Автоматизация может сделать снимки экранов устройств, а человек быстро проверит различия в изображениях, найденные алгоритмом. Человеческий разум слишком ценен, чтобы тратить его на то, с чем справится компьютер. Оставьте однообразные задачи машинам, человек нам еще пригодится.

— Опиши баг, который ускользнул от вас и заставил краснеть после выпуска.

Шелтон: Об этом обязательно нужно спрашивать? Скажите мне, хоть кто-нибудь выпускал идеальный продукт? К сожалению, не я! Самые болезненные для меня

баги всплыли после недостаточно тщательного тестирования изменений в конфигурации data-центра. Однажды новая версия была выпущена вообще без проверки, и качество результатов поиска ухудшилось. Зато мы узнали, насколько эти изменения важны для качества. С тех пор мы включили их в обязательную проверку. У нас есть набор автоматизированных тестов, которые должны быть выполнены до того, как изменения в конфигурации или данных пойдут в эксплуатацию.

— Как вы определили, какие тесты конфигураций нужно автоматизировать?

Шелтон: Проявили наблюдательность! Каждый раз, когда мы находили конфигурацию, отрицательно влиявшую на результаты поиска, мы писали тесты для нее и всех ее вариаций, которые могли бы показать такие же плохие результаты. Вскоре в нашем тестовом пакете сформировался целый набор проблемных конфигураций. Потом мы генерировали набор данных, на которых мы могли тестировать проблемные конфигурации. Этот тип багов стал встречаться реже благодаря нашим действиям. Вот хороший пример того, как автоматизация придает нам уверенности при выпуске изменений.

Интервью с директором разработки инженерных инструментов Ашишем Кумаром

Инструменты — вопрос жизни и смерти для Google. Ашиш Кумар — это человек, который отвечает за разработку инструментов. В его зоне ответственности находится весь чемодан внутренних инструментов Google: от IDE, в которых пишут разработчики, до систем код-ревью, от инструментов сборки, контроля исходного кода и статического анализа до общей тестовой инфраструктуры — за все отвечает он. Даже команды Selenium и WebDriver отчитываются перед Ашишем.

Мы встретились с Ашишем, чтобы узнать об этой части Google поподробнее.

— Область автоматизации в Google кажется чем-то магическим, во многом благодаря GTAC, а ты — человек, который за всем этим стоит. Приоткрой нам завесу тайны, какие возможности предоставляет твой инструментарий инженерам Google?

Ашиш: Команда разработки инженерных инструментов, а именно так мы называемся, отвечает за создание 90% инструментов, ежедневно используемых разработчиками Google, когда они пишут, собирают и выпускают качественные приложения. Последние 10% мы охватим, когда сможем поддерживать все команды, работающие с открытым кодом.

Google уникален тем, что огромное внимание уделяется созданию мощной и масштабируемой инфраструктуры для разработчиков. Люди извне обычно знакомы с технологиями MapReduce и BigTable, которые наши разработчики постоянно используют. Но наша внутренняя инфраструктура для разработки — это тоже большая часть нашей работы.

— **Можно конкретнее?**

Ашиш: Ладно, сами напросились! Набор инструментов включает:

- **Инструменты для работы с исходным кодом.** Они упрощают создание рабочих пространств, заливку изменений в код и соблюдение гайдлайнсов. Эти инструменты помогают просматривать сотни миллионов строк кода и дают удобный поиск для предотвращения дублирования. Они делают возможным индексирование и рефакторинг в облаке.
- **Инструменты разработки.** Это плагины для IDE, которые приспосабливают инструменты к коду Google и связывают их с нашими облачными сервисами. Эти инструменты помогают быстро и качественно рецензировать код благодаря возможности встроенных сигналов во время код-ревью.
- **Инфраструктура сборки.** Эта система позволяет нам распределить сборку мультиязычного кода по десяткам тысяч процессоров, используя такие объемы памяти и дискового пространства, что мне даже представить страшно! Система сборки работает как для интерактивного, так и для автоматизированного использования. Она формирует результаты за секунды, хотя та же работа в другом случае занимала бы часы.
- **Инфраструктура тестирования.** Это масштабная непрерывная интеграция. Это означает ежедневное выполнение миллионов тестовых пакетов по каждой заливке кода разработчиками. Наша цель — предоставить мгновенную (или почти мгновенную) обратную связь для каждого разработчика. У этого есть и другая сторона: нужно масштабировать веб-тестирование. Для тестирования продуктов Google каждый день запускаются сотни тысяч браузерных сессий с разными сочетаниями браузеров и платформ.
- **Инструменты локализации.** Их задача — постоянно переводить строки, специально выделенные разработчиками, чтобы локализованные версии наших продуктов выходили одновременно с англоязычными версиями.
- **Метрики, графики и отчеты.** Здесь речь об управлении багами по всем продуктам Google, сборке и хранению метрик разработки, тестирования и выпусков. Наша задача — предоставлять командам обратную связь, чтобы они могли улучшить свою работу.

— Значит, ты одновременно работаешь на всех фронтах. Чтобы достичь такого успеха, вы наверняка добавили много инноваций в рабочий процесс. Как вам удается сохранять баланс между новыми разработками и профильной работой? Ведь твоя команда не такая уж большая.

Ашиш: Все просто: мы не пытаемся взвалить на себя всю работу. Мы — централизованный отдел разработки инженерных инструментов. Команды часто разрабатывают специальные инструменты под свои задачи. Если какой-то инструмент начинают использовать и другие команды, мы оцениваем, можно ли включить его в общий инструментарий для всех сотрудников Google. Или, бывает, что мои инженеры сами предлагают какую-нибудь крутую штуку. Мы стараемся поддерживать все инициативы, но, конечно, у нас есть свои критерии для отбора инструментов перед тем, как сделать их централизованными. Первый критерий — инструмент должен потенциально сильно повлиять на производительность, второй — он должен быть полезен большому количеству разработчиков Google. В конце концов, мы работаем с общими инструментами, наше поле игры — это инструменты для широкой аудитории, которые полезны многим. Если инструмент полезен только одной команде — она сама им и занимается.

Мы много экспериментируем. Чтобы добиться большого успеха в следующем году, нужно работать уже сейчас, поэтому несколько небольших команд по одному-два человека всегда работают над экспериментами. Часто опыты проходят в «двадцатипроцентное» время, и я не против, так как это личное время инженеров, которым они вольны распоряжаться. Разумеется, часть экспериментов провалится, но те, которые приведут к успеху, с лихвой компенсируют любые неудачи. Мечтайте о большом, быстро понимайте, если идете не туда, и не сдавайтесь!

Некоторые инструментальные проекты не самодостаточны, поэтому сложно измерить их влияние, но они все должны давать ощутимый толчок в сторону увеличения производительности Google.

— А была ли идея, которая казалась тебе неудачной, но привела к успеху?

Ашиш: Да! Масштабная непрерывная интеграция. Она казалась недостижимой, потому что требовала огромного объема работы. Тогда у нас были тысячи машин, которые непрерывно выполняли циклы интеграции для каждого проекта. Кто-то предложил создать инфраструктуру, которая сделала бы этот процесс централизованным для всего Google. Такая инфраструктура опрашивала бы системы управления кодом на предмет изменений, держала бы в памяти огромный граф мультиязычных зависимостей этих изменений, а потом автоматически собирала бы и запускала нужные тесты. Многие скептически относились к проекту, так как идея была слишком масштабна и наши серверы могли не потянуть. Я тоже был среди скептиков: решение требовало огромного количества ресурсов. Но шаг за шагом наши инженеры понемногу преодолевали технические трудности до тех пор, пока не добились успеха: система запущена и работает. Собственно, это и есть

схема нашей работы с проектами: начинаем с малого, а если практическая польза и потенциал доказаны, разворачиваемся на полную.

— А есть ли инструмент, который обещал быть успешным, но провалился?

Ашиш: Снова да! Удаленное парное программирование. В Google много распределенных команд. Многие команды применяют парное программирование и другие гибкие методы разработки. Часто бывает, что вы работаете над кодом, который написал человек из другого офиса, и если у вас появляются вопросы, то возникает задержка, влияющая на производительность.

Мы хотели построить плагин для удаленного парного программирования. Мы планировали написать инструмент, встроенный прямо в среду разработки, который бы давал возможность связаться с автором кода через Google Talk. В идеале автор кода получал доступ к рабочему пространству, и ребята могли править код вместе, наблюдая друг за другом по видео. Это парное программирование, только без присутствия.

К сожалению, опробовав тестовую версию только с простым совместным редактором без интеграции с Google Talk, мы свернули проект. Статистика использования не дала ожидаемых результатов, показала, что разработчики не заинтересованы в продукте. Может быть, мы переоценили важность проблемы.

— Твой совет компании, которая хочет построить непрерывный процесс автоматизации? С каких инструментов стоит начать?

Ашиш: Самое важное — создать такую среду разработки, чтобы в ней даже разработчик-новичок смог работать с вашей командой. Должно быть невероятно просто взять код из репозитория, отредактировать, протестировать, отладить его и опубликовать. Если вы сформируете настолько удобную среду, то все ваши разработчики будут работать более продуктивно и вы сможете выпускать ваше ПО без задержек.

Как создать такую среду? Нужно четко определить зависимости, сделать их явными и создать систему непрерывной интеграции, которая делает свое дело. Главное, чтобы она быстро предоставляла информацию разработчикам. Если на получение обратной связи уходит больше пары минут, добавьте еще машин. Время процессора стоит гораздо дешевле рабочего времени инженера. Запустить, отладить или развернуть код должно быть так же просто, как ввести команду. Если вы работаете в веб-компании, упростите процедуру частичного развертывания.

— Какие инженеры нужны в твоей команде? Вряд ли любой разработчик сможет разрабатывать инструменты?

Ашиш: Разработка инструментов предполагает особую любовь к компьютерным наукам. Я говорю о разработке языков и компиляторов, системном программировании и т. д. Мне нужны такие инженеры, которые воспринимают разработчиков

как потенциальных пользователей и ловят кайф от того, что их инструменты помогают другим разработчикам приносить больше пользы.

— Раз уж мы заговорили о пользователях, как ты убеждаешь людей применять твои инструменты?

Ашиш: Разработчики Google — это очень благодарные пользователи. Раз в неделю мы проводим встречи и демонстрируем наши инструменты. Инженеры приходят, задают вопросы, и если инструмент решает их задачи, они берут его на пробу. На инструменты, которые решают насущные проблемы, довольно большой спрос. Чем ближе к реальности, тем эффективнее. Надо быстро отказываться от проектов, которые нерезультативны или не востребованы.

— А тебе когда-нибудь попадались инструменты, которые только мешали работе или приносили больше вреда, чем пользы?

Ашиш: Да, но я даже не запоминаю их, так как все подобные проекты очень быстро забрасываются. Цель разработки инструментов — автоматизация процесса и его упрощение. Не нужно автоматизировать неправильные решения. Если разработчик совершает ошибку, зачем упрощать ему этот процесс? Отступите на шаг и оцените: может быть, нужно заняться чем-то более полезным.

— Над чем сейчас работаешь? Какие новые инструменты готовит твоя команда?

Ашиш: Мне приходится много работать над тем, чтобы просто «не отставать». Всё меняется так быстро, что наши веб-инструменты постоянно требуют доработки. Иногда новые условия заставляют нас переделать инструмент, а иногда помогают найти совершенно новую функциональность. Изменения — постоянное испытание и открытие. Многое из того, что мы делаем, связано с внутренними проектами, поэтому я не имею права их разглашать. Могу только поделиться с вами важным выводом: масштабирование, масштабирование и снова масштабирование.

Интервью с Суджаем Сани, директором по тестированию в индийском Google

Мы привлекаем талантливых людей в разных странах через распределенную систему офисов, создавая региональные и глобальные центры разработки Google. В Индии было много способных инженеров, поэтому наш первый глобальный центр тестирования

появился именно там, в Хайдарабаде. Сотрудники центра работают над ключевыми продуктами Google, и это помогает переходу от ручного тестирования (служба тестирования, вы помните?) к инженерному тестированию. Суджай Сани — директор по тестированию, который основал и возглавил направление продуктивности разработки в Индии.

— Индия находится далеко от главного офиса Google в Маунтин-Вью. Почему именно здесь появился столь важный технический отдел?

Суджай: Наш отдел формировался так же, как и любой другой в Google, команда собралась там, где была концентрация подходящих специалистов. Ключевой фактор появления отдела именно в Индии — не низкая стоимость рабочей силы, а наличие исключительных специалистов в этом регионе. Сейчас Индия — один из многих региональных центров, где работают и разработчики, и тестировщики, такой же, как в Лондоне, Нью-Йорке, Киркленде, Бангалоре и других городах.

Из регионального офиса мы выросли в глобальный центр. Сейчас Хайдарабад — это центр Азиатско-Тихоокеанского региона Google, офис в Цюрихе — европейского, а в Нью-Йорке — центр Восточного побережья США. Центры объединяют и организовывают работу более мелких офисов Google в своем регионе, они помогают экономить время и распределять ресурсы.

Хайдарабад стал не просто глобальным центром, но и кузницей кадров для всего Google. Многие инженерные решения для тестирования начинались именно здесь. Ребята из этого офиса работали над ключевыми проектами Google, что ускорило переход тестирования как службы к направлению продуктивности разработки.

— Какую роль сыграл индийский офис в эволюции тестирования Google?

Суджай: Как я уже говорил, наш региональный центр стал первым. Хотя мы открыли офис в Бангалоре, хайдарабадский центр (или сокращенно HYD) быстро стал глобальным хабом инженерных подходов к тестированию. С самого начала в HYD работали и инженеры по тестированию, и разработчики в тестировании, и большое количество временных и внешних сотрудников. Они занимались самыми значительными продуктами Google: Search, Ads, Mobile, Gmail, Toolbar и другими. В первую очередь они разработали инфраструктуру тестирования и фреймворк для автоматизации тестирования, чем сильно ускорили выпуск продуктов. В 2006–2007 годах в команде HYD работала примерно половина всех разработчиков в тестировании в Google. Говорят, что роль разработчика в тестировании укрепилась в результате усилий первого тестировщика, нанятого в HYD. Не знаю, правда это или нет, но мы сделали возможной трансформацию службы тестирования в направление продуктивности разработки.

К концу 2007 года мы бросили все силы на развитие команды в новых областях, уменьшение фрагментации команд и создание зрелого сообщества сильных инженеров, чтобы справиться с растущим количеством новичков. В начале

2008 года мы сами стали открывать региональные центры тестирования, чтобы команды разработки могли работать бок о бок с локальными командами тестирования. Теперь мы могли начать работать с до сих пор нетронутыми областями. У нас дошли руки до продвинутых средств обнаружения задержек, инструментов контроля производительности и стабильности клиент-серверных и облачных систем, механизмов выявления деградации и средств клиентского тестирования.

Тогда же произошло еще одно изменение — мы начали вкладывать время и ресурсы в облачное тестирование и инфраструктуру инженерных инструментов. Мы работали над проектами Cloud Code Coverage, Cloud Scalable Test Infrastructure, Google Toolbox, разными IDE и другими экспериментами, большинство из которых в итоге стали боевыми инструментами. Наша команда не только разрабатывала нужные инструменты для инженеров Google, но и создавала основную инфраструктуру для внешних разработчиков, работающих с открытым кодом. Инженеры из HYD работали над App Engine, Selenium, плагинами Eclipse, IntelliJ и другими проектами опенсорс-сообщества.

— Все это хорошие, важные проекты. А ты можешь привести пример проекта, целиком выполненного в HYD?

Суджай: Да. Например, программа Google Diagnostic Utility была полностью разработана в Хайдарабаде. С ее помощью наша служба поддержки работает с пользователями по диагностике проблем, которые у них возникают с продуктами Google. Эта программа упрощает получение технических подробностей систем и железа пользователя.

Есть и другие примеры. Наш отдел сосредоточен на разработке инженерной инфраструктуры, инструментов и тестов для всего Google. Для разработчиков мы создаем среду разработки, основную облачную инфраструктуру для компилирования, тестирования и измерения покрытия кода, делаем возможным любой статический анализ. Для тестировщиков мы разрабатываем систему для анализа нагрузки и производительности облачных сервисов Google, а еще создаем инструменты тестирования для таких важных продуктов, как Search, Enterprise, Gmail и Ads.

— Давай поговорим об этих инструментах. Даже названия звучат интересно. Ты упомянул инструмент, который измеряет покрытие кода, расскажи о нем подробнее. Эта тема очень популярна в блоге Google Testing.

Суджай: Покрытие кода — это метрика, которая показывает эффективность тестов для кода проекта. В традиционном подходе каждая команда должна выделить отдельные ресурсы (инженерные, аппаратные и программные) для изменения этого покрытия. А в Google есть специальная группа людей, находящаяся в Индии, которая следит за тем, чтобы все инженерные команды легко получали свои метрики покрытия кода. Для этого команда должна потратить всего пять минут, чтобы один раз настроить эту функциональность. После настройки у них

будут все метрики проектов и сборок с централизованной системой просмотра и анализа отчетов.

Измерение покрытия поддерживается для тысяч проектов, всех основных языков программирования и миллионов файлов с исходным кодом. Инфраструктура покрытия интегрирована с облачной инфраструктурой Google для компилирования и сборки кода. Она масштабируется для постоянных изменений кода (происходящих ежеминутно!) и десятков тысяч сборок в день. Инфраструктура способна меняться вместе со стремительно растущей кодовой базой Google.

Еще в нашем арсенале есть вспомогательная система для назначения тестам приоритетов. С ее помощью определяются тесты, которые должны выполняться для конкретных изменений в коде. Это улучшает покрытие, прибавляет уверенности в качестве кода и ускоряет обратную связь. Тем самым мы экономим Google инженерные ресурсы.

— Похоже, это правильный подход к покрытию кода. А теперь расскажи о приложении Diagnostic Utility, которое ты упоминал.

Суджай: Diagnostic Utility задумали и реализовали разработчики в тестировании нашего центра в свое «двадцатипроцентное» время. Технических знаний обычного пользователя часто не хватает, чтобы дать нужную информацию для диагностики и отладки проблемы, а это приложение автоматически предоставляет эти данные.

Чтобы разобраться в проблеме пользователя, иногда нужны определенные технические данные. Это могут быть простые данные, которые клиент может предоставить (например, версия ОС), но бывает, что требуются подробности о версиях и конфигурациях приложений, поиск которых ставит пользователя в тупик.

Приложение Diagnostic Utility решает эту проблему. Теперь, когда службе поддержки нужна дополнительная информация о системе пользователя, они создают новую конфигурацию для этой утилиты, где перечисляют, какая информация требуется. Дальше пользователь получает по почте или скачивает с сайта поддержки небольшой (около 300 Кбайт) исполняемый файл, подписанный сертификатом Google. Этот файл диагностирует машину пользователя, собирает указанные в конфигурации данные и показывает их пользователю, чтобы он подтвердил их отправку в Google. Приложение хорошо воспитано — после отправки данных оно прибирает за собой и удаляется. Мы заботимся о конфиденциальности, поэтому все собранные данные отправляются только с согласия пользователя.

Служба поддержки пересыпает данные разработчику для отладки. Приложение Diagnostic Utility делает работу команд клиентских приложений (например, Google Chrome и Google Toolbar) проще, а самому пользователю облегчает процесс получения технической поддержки.

— Ты пару раз упоминал тестирование нагрузки и производительности. Что здесь интересного расскажешь? Насколько мы понимаем, ваша команда активно участвует в тестировании производительности Gmail?

Суджай: Google выпускает очень много веб-приложений, и очень важно обеспечить быстрое взаимодействие пользователей с ними. А раз так, то тестирование производительности клиентской части — анализ скорости выполнения JavaScript и отображения страниц — обязательно должно проводиться перед каждым выпуском. Раньше на выявление причин задержек и последующую отладку могли уходить дни, а то и месяцы. Ребята из индийского отделения продуктивности разработки построили инфраструктуру для тестирования производительности фронтенда Gmail, чтобы покрывать самые важные действия пользователя. То, что пользователи делают чаще всего, должно проходить через тщательное тестирование производительности. Мы настраиваем специальные серверные конфигурации, тесты выполняются в контролируемой среде, которая минимизирует отклонения и помогает выявлять ухудшения в работе системы.

Наше решение состоит из трех частей:

- **Предварительные тесты.** Инженеры могут проводить тесты и измерять задержки перед отправкой изменений в коде в репозиторий. Это ускоряет обратную связь и снижает вероятность попадания багов в кодовую базу.
- **Непрерывная сборка.** Серверы тестирования синхронизируют последние изменения кода и тут же запускают соответствующие тесты, чтобы заметить и перехватить регрессию. Мы уменьшили время на обнаружение регрессии до нескольких часов или даже минут.
- **Анализ задержек.** Инфраструктура выявляет изменения, из-за которых появилась задержка. Для этого весь комплект изменений делится на части, и тесты проводятся в контрольных точках.

Это решение уже помогло найти много критических багов до выпуска и сильно повысило качество продуктов. К тому же разработчики могут сами запускать тесты.

— **Расскажи о каких-нибудь экспериментах. Чему вы научились на таких проектах?**

Суджай: Мы экспериментировали с инструментами обратной связи, которые собирают нужные данные и предоставляют метрики командам, чтобы сделать их работу еще более эффективной. Сюда же входят инструменты визуализации кода, измерения сложности кода и другие подобные инструменты. Еще мы работали с расширенной средой разработки, которая совершенствовала привычные механизмы использования среды и метрик, чтобы повысить качество кода и пропускную способность команды. Еще был инструмент, который собирал информацию после выпуска продукта, чтобы разработчики могли принять правильные меры.

— Напоследок расскажи, что ты думаешь о глобальной организации тестирования. Ты ведь работаешь в компании, где разработка сильно распределена.

Суджай: Это непросто сделать, но мы показали, что это может работать. Я вынес несколько важных уроков.

Модель «следуй за солнцем»¹ хорошо работает, если правильные команды занимаются правильными проектами. Наша раскиданная по всему миру команда смогла справиться с испытаниями, пусть и не без ошибок. Очень важно иметь хороший процесс передачи работы между часовыми поясами. Тщательно выбирайте людей и проекты. Вам нужны хорошие командные игроки, неравнодушные к продуктам, которые вы создаете.

Нас очень выручило краудсорс-тестирование. Мы пользуемся огромным резервом талантливых специалистов в Индии. Разница во времени нам здесь нестрашна, и это действительно работает.

Чтобы победить, нужны талантливые специалисты, которым можно доверить ключевые проекты. Повторю, Индия была выбрана не потому, что Google хотел сэкономить на рабочей силе, — есть места и подешевле. Просто мы всегда старались поддерживать качество найма и условия работы на высоком уровне. Мы вносим большой вклад в Google, а наши сотрудники получают достойную карьеру. Все в выигрыше.

Интервью с тест-менеджером Брэдом Грином

Брэд Грин, которого прозвали Босоногим, был тест-менеджером многих продуктов Google, включая Gmail, Google Docs и Google+. Сейчас он менеджер в разработке Google Feedback и еще экспериментирует с фреймворками веб-разработки в проекте Angular. Он известен тем, что у него много гениальных идей и нет обуви.

— Мы знаем, что раньше ты был разработчиком в компании Apple. Что заставило тебя перейти в Google и почему именно в направление продуктивности разработки?

Брэд: Линус Апсон помог мне решиться. Мы работали с ним в NeXT², и он был воодушевлен компанией Google. Собеседования шли полгода, и в какой-то момент Патрик Коупленд заманил меня в направление продуктивности разработки. Фо-

¹ «Следуй за солнцем» (от англ. — follow the sun) — модель рабочего графика межнациональных компаний, при котором руководитель, находящийся на одном континенте, в конце своего рабочего дня связывается с руководителем на другом континенте (где день только начинается) и передает ему дела, а тот в конце своего рабочего дня передает ему их обратно. Таким образом работа не прерывается, а просто двигается «вслед за солнцем».

² NeXT — американская компания, основанная в 1985 году Стивом Джобсом. Занималась разработкой платформ для вузов и бизнеса. В 1996 году была куплена Apple.

кус удалось: я узнал в этом отделе так много, что теперь гораздо лучше подхожу на должность менеджера по разработке, на которую я вернулся. Пожалуй, каждого разработчика стоит отправлять на стажировку в команду тестирования.

— Что тебя больше всего удивило в культуре тестирования Google в начале твоей работы?

Брэд: Я пришел в Google в начале 2007 года, и изменения, о которых Патрик говорил в своем предисловии, еще не завершились. Меня поразило, как много ноу-хау тестирования здесь использовали. В любой новой команде я находил эксперта по тестированию, который меня удивлял. Проблема была в том, что весь этот богатый опыт был неуправляем. У эскимосов есть сотни слов для обозначения снега¹, а в Google были десятки терминов для одного вида теста. Когда я присоединялся к команде, мне приходилось учить их язык тестирования и термины. В одних командах процессы были жестко формализованы, а в других нет. Нужно было что-то менять.

— Что больше всего изменилось в тестировании с момента твоего прихода в Google?

Брэд: Две вещи. Первая: средний разработчик сейчас больше участвует в процессе тестирования и автоматизации. Он знает, что такое тесты API, что такое модульные, интеграционные и системные тесты. Он инстинктивно создает больше тестов, если сталкивается с проблемами во время непрерывной сборки. Ему даже не нужно напоминать об этом. Конечно, это существенно улучшило исходное качество и увеличило скорость выпуска. Вторая: нам удалось привлечь сотни первоклассных разработчиков на роли, связанные с тестированием. Я думаю, эти две вещи связаны. В Google появилась культура, в которой тестирование имеет значение, и тестировать — это круто.

— Расскажи, что значит быть менеджером в Google? Что самое трудное, самое простое и самое интересное в работе менеджера в Google?

Брэд: Самое сложное — научиться управлять людьми, которые могут руководить своей работой сами. Вариант «делай так, как я сказал» сработает разок, а потом вы потеряете контакт с этими талантливыми ребятами, и они будут делать только то, что, по их мнению, лучше для проекта. Моя работа приносила максимальную пользу, когда я помогал, а не руководил, когда я делился информацией и создавал комфортные условия для моих сотрудников, чтобы они занимались любимым делом. Если мне приходилось приказывать, я чувствовал, что не развиваю ребят, не помогаю им учиться принимать решения. Надо стать не названным, а признанным лидером среди чрезвычайно умных и увлеченных инженеров. Менеджеры, будьте осторожны!

¹ Как оказалось, это неправда: http://ru.wikipedia.org/wiki/%D0%A5%D0%BA%D0%BC%D0%B8%D0%BC%D0%BE%D0%BD%D0%BE%D1%81%D1%82%D0%B8%D0%BC%D0%BE%D0%BD%D0%BE%D1%81%D1%82%D0%B8_%D0%BD%D0%BD%D0%BE%D0%BB%D0%BE%D0%BD%D0%BE%D1%81%D1%82%D0%B8.

— Твоя команда провела масштабную разработку метрик для разработчиков, которые тестируют. Какие метрики вы используете? Какие данные отслеживаете? Как это влияет на качество?

Брэд: Признаюсь, мы двигаемся два шага вперед, один назад. Я терплю неудачи в этой области уже четыре года — и многому научился! Я говорю «неудачи», потому что мы проделали колоссальный путь в поисках волшебных метрик для качества кода и тестов, которые могла бы использовать любая команда. Метрики трудно обобщить, потому что контекст очень важен. Да, чем больше тестов — тем лучше, если только они не медленные и не ненадежные. Да, малые тесты лучше больших, если только вам не нужно проверить строение всей системы. Измерения полезны, но реализация каждого теста уникальна и, наверное, так же близка к искусству, как и код самого приложения.

Оказывается, есть социальные факторы тестирования, которые гораздо сложнее, чем любые технические. Все понимают, что нужны хорошие тесты, но многим менеджерам трудно заставить команды их писать. Лучшее средство для мотивации — соревнования. Хотите получить больше малых тестов? Сравните свою команду с другой. Эй, у той команды тестовое покрытие — 84%, а полный набор тестов выполняется за пять минут! И мы позволим им обыграть нас?!

Измеряйте все, что плохо лежит, но доверяйте людям. При этом все равно будьте реалистом: можно сколь угодно тщательно тестировать продукт внутри компании, но он все равно окажется в руках пользователя. Во внешней среде работают другие законы, вы не можете их ни предсказать, ни воспроизвести.

— Это объясняет твое участие в Google Feedback. Можешь немного рассказать о нем? Какие проблемы должен решать этот продукт?

Брэд: Feedback помогает пользователям сообщить о проблемах, которые они находят в продуктах Google. Звучит несложно? Просто прикрутить к странице форму отправки данных и дать пользователю проставить галочки? Много команд шло этим путем, а потом они не справлялись с лавинами отчетов — часто их приходило по несколько тысяч в день. Еще возникали трудности с отладкой, потому что информация от пользователей не всегда была полной и точной. Для решения этих проблем был создан Feedback.

— Можешь рассказать, как работает Google Feedback?

Брэд: Feedback собирает всю возможную информацию, соблюдая конфиденциальность пользователя. Браузер, операционная система, плагины — все эти данные очень важны для отладки и легко собираются программой. Особенная хитрость со скриншотами. В целях безопасности браузеры не копируют изображение своего содержимого. Нам пришлось заново реализовать механизм визуализации на JavaScript безопасным путем. Теперь мы делаем скриншот и просим пользователя выделить на нем проблемную область. Потом мы просим его описать проблему

текстом. Невозможно научить пользователей создавать идеальные баг-репорты, но со скриншотами расплывчатые описания становятся яснее.

— Разве с таким количеством пользователей вы не будете получать сообщения об одном и том же баге?

Брэд: Чтобы избежать дублирования багов, мы автоматически группируем похожие сообщения. Если тысяча пользователей сообщает об одной проблеме, мы объединяем эти сообщения в один слот. Вручную обработать такое количество сообщений было бы невозможно. Затем группы ранжируются, и мы определяем, какие проблемы затрагивают большее количество пользователей. Их мы и будем решать в первую очередь.

— Сколько человек в команде Google Feedback?

Брэд: В команде 12 разработчиков и три менеджера. Последних, конечно, больше, чем обычно, но здесь это оправданно, так как нужно координировать много проектов, связанных с Google Feedback.

— Какие самые серьезные проблемы возникали при запуске Google Feedback?

Брэд: Технически снятие скриншота было невероятно сложной задачей. Многие думали, что даже браться за такое — безумие. А сейчас все функционирует отлично. Мы справились с тем, что баг-репорты создаются на разных языках, но нам предстоит еще много работы. Автоматизированная группировка проблем была и остается очень сложной задачей.

— Какое будущее ждет Google Feedback? Возможно ли, что когда-нибудь система станет доступной не только для сайтов Google?

Брэд: Наша цель — дать пользователям возможность общаться с нами о проблемах в наших продуктах. Сейчас это монолог. Я думаю, что в будущем нам удастся создать инструменты для диалога. Мы не планировали выпускать наш продукт в большой мир, но сейчас мне кажется, что это хорошая идея.

— Каким ты видишь следующий шаг в развитии тестирования программных продуктов?

Брэд: Мне бы хотелось увидеть среду разработки, в которой тестирование — первостепенная фича, а не прикрученная позже. Представьте, что языки, библиотеки, фреймворки и инструменты сами знают, какие тесты будут нужны вашему коду, и помогут вам их написать. Это будет круто. А пока нам придется прилеплять тестовые фреймворки к разработке. Тесты трудно писать и тяжело сопровождать, они нестабильны во время выполнения. Я думаю, что если мы будем выпекать наши тесты на самом низком уровне, это принесет нам много плюшек.

— У тебя есть какой-нибудь компромат на доктора Джеймса Уиттакера, о котором ты бы хотел поведать миру?

Брэд: Кроме того инцидента с костюмом бедняжки Мэри? Пожалуй, нет. Мы, менеджеры, своих не сдаем!

Интервью с Джеймсом Уиттакером

В Google приход Джеймса был встречен фанфарами, и он сразу стал одним из самых известных людей в нашем тестировании. Он обладает феноменальной харизмой, и все, что он делает, привлекает повышенное внимание: его выступления на GTAC собирают толпы, посты Джеймса в нашем блоге становятся самыми популярными. Он покорил наши офисы в Сиэтле и Кирклэнде, а по уровню влияния на всю компанию Google он приближается к Патрику Коупленду. Конечно, Пат — формальный руководитель, но у нас есть еще один неформальный лидер в области тестирования — Джеймс Уиттакер.

Джайсон Арбон и Джефф Каролло поговорили с Джеймсом в его офисе.

— В 2009 году ты перешел в Google из Microsoft. Когда ты объявил в своем блоге о том, что уходишь из Microsoft, почему ты не назвал новое место работы? Зачем эта таинственность?

Джеймс: Сразу такой жесткий вопрос? Ого! Мне обещали, что будет просто!

— А ты обещал ответить на наши вопросы, так что давай!

Джеймс: Это был самый простой способ оповестить максимальное количество людей одновременно. Тогда Twitter еще был не очень популярен, и я воспользовался блогом MSDN. Оказывается, коллеги чаще читали мой блог, чем мои электронные письма. Уход из Microsoft немного пугал, и я хотел высказаться один раз, но громко и доступно, чтобы избежать сотни прощальных встреч. Так что это было лучшее решение.

Люди, которые были в курсе моего увольнения, потратили уйму времени, отговаривая меня. Тяжело было уходить из компании, в которой нравилось работать. Трудно было расставаться с людьми, с которыми работал вместе годами. Я чувствовал себя ужасно в связи с уходом, и мне не хотелось повторно обдумывать мое решение. Мне нравится Microsoft, и я уважаю инженеров, которые здесь работают. На самом деле меня действительно могли отговорить от перехода, но я очень хотел работать в Google и в то же время не хотел давать коллегам возможности повлиять на мое решение.

— Почему? Что такого в Google так сильно тебя привлекало?

Джеймс: Знаешь, это может прозвучать странно. Я начинал как профессор-консультант, потом открыл свой бизнес и успел попробовать все, кроме работы

в большой компании. И раз уж я решился на работу в большой компании, пусть она будет самой большой! Чем больше, тем лучше! Чем круче работа, тем лучше! Чем больше аудитория, тем интереснее! Я хотел понять, насколько успешным я могу стать в отрасли, так почему бы не проверить это в лидирующей компании? Так я оказался в Microsoft, а потом в Google. Я хочу работать в больших компаниях и выбираю из них лучшие.

Но самый сильный аргумент — это репутация Google как компании с самой крутой организацией тестирования. Это меня покорило. Долгое время это звание принадлежало Microsoft, но Патрику Коупленду удалось его отбить. Я решил, что самое интересное место работы для тестировщика — это Google.

Собеседования только подкрепили мое мнение. Я встречался с Патриком Коуплендом, Альберто Савоя, Брэдом Грином, Шелтоном Маром, Марком Стрибеком и многими другими. Наши разговоры были потрясающими. Мы с Альберто изрисовали целую доску во время интервью. Только потом он заметил, что забыл задать мне необходимые для собеседования вопросы. С Шелтоном мы разошлись во взглядах, но он настолько уважительно принял мое мнение, что это произвело на меня большое впечатление. Сейчас мы соглашаемся с ним гораздо чаще! А Брэд? Он был крут. Во время собеседования на нем не было обуви (а это был февраль), и это только добавило нашему разговору непринужденности. Марк большую часть собеседования уламывал меня перейти в Google. Атмосфера просто-таки фонтировала идеями. У меня голова шла кругом.

Признаюсь, после всех собеседований я был утомлен. Помню, я упал в свою машину и думал, что нашел лучшую компанию, но хватит ли мне сил, чтобы здесь работать? Я не был уверен, что смогу внести ощутимый вклад в общее дело. Это все пугало меня, это все было за пределами моей зоны комфорта. Но я люблю сложные задачи, и дух соревнования пересилил мой страх. Кому нужна легкая работа?

— И мы оправдали свою репутацию?

Джеймс: О да, работа оказалась непростой. Но я думаю, вопрос скорее об увлеченности? Честно говоря, и в Microsoft хватает умных и увлеченных тестированием людей. Разница в том, что в Google проще реализовывать свои увлечения. Мы с Альберто никогда не числились в одной команде, но легко работали вместе в «двадцатипроцентное» время. С Брэдом, например, мы до сих пор работаем вместе над средой разработки и автоматизацией баг-репортов (Брэд через Google Feedback, я через BITE). Google хороший тем, что дает возможности для такого сотрудничества и поощряет это.

— Мы работаем с тобой в Киркленде и заметили, насколько отличается темп твоей здешней команды и ее общий настрой. В чем твой секрет?

Джеймс: Да, сейчас в Киркленде дела идут гораздо лучше, но это не только моя заслуга. Частично дело в критической массе правильных людей. Как раз перед

моим приходом наняли много талантливых специалистов, и за первые несколько кварталов наш штат тестировщиков стал в четыре раза больше. Я смог строить большие команды и объединять людей, занимающихся похожей работой, даже если они работали над разными продуктами. Теперь вместо одного тестировщика в команде разработки появилась группа специалистов, которые сотрудничали друг с другом и делились идеями. Такая схема творит чудеса и с настроением команды, и с ее производительностью.

Так как у меня было достаточно людей, я смог снять ветеранов вроде вас двоих с текущих проектов, заняв более серьезными задачами. Джейф тогда занимался реализацией очереди предварительной отправки для Google Toolbar, а Джейсон тестировал Google Desktop. Какое глупое использование ваших талантов! Пode-люсь секретом хорошего менеджера: подберите каждому человеку идеальную для него задачу — и ваша работа сделана. Люди счастливы, и проекты делаются лучше. Правда, эта роскошь стала доступной только благодаря тому, что к моему приходу набралась критическая масса сотрудников.

Еще один плюс большого количества сотрудников — стало больше людей, которые могут тратить свое «двадцатипроцентное время» на эксперименты. Мне удалось собрать команды на несколько рискованных проектов. Мы начали с инструментов и опытов, которые не имели отношения к выпуску программных продуктов, скорее мы просто делали то, что нам было интересно. Оказалось, что ничто так не воодушевляет тестировщиков, как разработка инструментов. Честно говоря, наверное, это самая приятная часть работы. Наверное, в душе я больше разработчик инструментов, чем тестировщик.

— Что тебе больше всего нравится в организации работы Google?

Джеймс: Командная игра. Я твержу об этом кандидатам, когда хочу убедить их работать у нас: тестировщики рапортуют тестировщикам и сами определяют свою судьбу — это то, что я больше всего люблю в Google. Тестировщики сами нанимают сотрудников, сами проводят собеседования, сами решают, кого повышать. ООН признала нашу Республику Тестирование независимым государством!

В нашей стране нет формального подчинения. И еще одна тому причина — дефицит тестировщиков. Ни одна команда разработки не получит себе тестировщика просто так, его нужно заработать. И тестировщикам приходится постоянно изобретать короткие пути. Нас настолько мало, что нам нужно все хорошо приоритизировать. Нам нужно постоянно улучшать автоматизацию, учиться договариваться с разработчиками. Дефицит способствует оптимизации. Пат многое сделал правильно, но дефицит, на мой взгляд, это одна из вещей, которые сильно повлияли на культуру.

— Долго ты привыкал к культуре Google? Ведь ты пришел из Microsoft, где не было централизованной организации тестирования?

Джеймс: В начале моей работы Пат Коупленд дал мне два совета. Первый — просто изучи обстановку. Это было критически важно. Нужно время, чтобы разобраться в том, как работает большая компания. Чтобы успешно работать в Google, нужен другой набор навыков, не такой, как в Microsoft. Так что первые пару месяцев я следил совету Пата — только слушал и задавал вопросы. Кажется, я даже продлил себе период адаптации на пару недель.

— А второй совет?

Джеймс: Да, извини, увлекся. Иногда Пат говорит умные вещи, и, кажется, мне попалась как раз такая. Второй совет мне совсем не понравился, но оказался еще полезнее первого. Пат отвел меня в сторонку и сказал: «Слушай, парень, я знаю, что у тебя есть репутация за пределами Google, но внутри ты еще ничего не сделал». Пат — человек прямой, и в разговоре его сложно понять неправильно. И я осознал — Google не было дела до того, чем я занимался до прихода сюда. Чтобы состояться именно здесь, я должен сделать что-то внутри компании. Нельзя заслужить уважение в Google, просто прогуливаясь и насвистывая. Пат предложил мне взяться за большой проект и удачно завершить его, да еще и так, чтобы это было заметно. Я выбрал Chrome и Chrome OS и стал там первым менеджером по тестированию. Проект был выпущен, и я передал его одному из своих ребят.

Пат был прав. Менять порядки стало легче после того, как я сделал что-то значительное. Мое резюме позволило мне попасть в компанию, но в стенах Google еще нужно удержаться. Я справился с задачей и внес свой вклад в продукт, важный для многих людей, поэтому ко мне стали прислушиваться. Если я когда-нибудь поменяю место работы, то снова воспользуюсь этой формулой: сначала все узнать, потом заслужить репутацию и только потом — искать возможности для нововведений.

— Кроме тестирования продуктов, какими направлениями ты занимался по совету Пата?

Джеймс: Да, он попросил меня взять под крыло роль инженера по тестированию. Тогда профессия разработчика в тестировании уже укрепилась, и мы понимали, как строится их карьерная лестница. Мы понимали ожидания людей и знали, как измерять их прогресс и как повышать. А вот с тонкостями роли инженера по тестированию нужно было разбираться. Пат спланировал вернуть фокус на эту роль как раз к моему приходу. Думаю, что Пат заранее хотел дать мне эту задачу, вряд ли это совпадение. Подозреваю, что ему казалось, что весы склоняются в сторону разработчиков в тестировании, и он хотел вдохнуть новую жизнь в роль тестировщика. Конечно, он мне ничего такого не говорил, это просто мои догадки.

— И что же ты сделал для инженеров по тестированию?

Джеймс: Мы с Патом создали рабочую группу инженеров по тестированию, которая существует до сих пор. Сначала мы собирались раз в неделю на два часа, постепенно свели к одной часовой встрече в месяц. Пат приходил на первые встречи, а потом оставил меня рулить этим. Команда состояла из двенадцати инженеров по тестированию, отобранных Патом лично. Я еще был новичком и никого из них не знал. На первой встрече мы создали два списка: что в нашей роли круто и что — отстой. Составление списков стало моментом истины для многих участников. Они были единодушны в своем восприятии плохих и хороших факторов в профессии. Меня поразило, насколько они были честны с Патом, — никто даже не пытался склонять острые углы! Я видел уйму встреч, на которых все ждали, пока высажется самый авторитетный участник, или отмалчивались из-за его присутствия. В Google все было иначе. Никого не волновало, что думал Пат, — это была *их* встреча, и если Пата что-то не устраивало, это была его проблема. Все это сильно воодушевило меня.

Рабочая команда много сделала для определения роли инженера по тестированию. Например, мы переписали рекомендации по карьерному продвижению. Мы выдвинули новый вариант карьерной лестницы на открытое голосование всего сообщества инженеров по тестированию в Google, и ее приняли. Это было здорово — я даже выписал премию всей рабочей группе, чтобы отпраздновать это событие. Наши идеи исходили «снизу», от людей, работающих в специальности, и поэтому принимались легко. Мы написали рекомендации по проведению интервью и разослали их рекруттерам и инженерам, участвующим в собеседованиях. Я думаю, что сейчас роль инженеров по тестированию определена так же четко, как и роль разработчика в тестировании.

— Ты уже достаточно давно работаешь в Google и знаешь всю подноготную. Можешь выдать нам секретный ингредиент Google? Что делает наше тестирование магическим?

Джеймс: Рецепт такой: возьмите высокую квалификацию тестировщиков, добавьте дефицит человеческого ресурса (который заставляет разработчиков помогать, а тестировщиков — оптимизировать), первым делом автоматизируйте (чтобы люди делали только то, в чем компьютеры не сильны), введите способность работать итеративно и быстро внедрять изменения после обратной связи от пользователей. Смешать, но не взбалтывать! Любая компания, которая захочет воспроизвести нашу организацию тестирования, должна начать с этих четырех ингредиентов.

— Как насчет другой книги? У тебя уже задумана следующая книга по тестированию?

Джеймс: Не знаю, я ведь никогда не планирую книги заранее. Моя первая книга родилась из курсовых заметок, которые я использовал для преподавания тестиро-

вания в Политехническом университете Флориды. Однажды после презентации на конференции STAR ко мне подошла женщина и спросила, не думал ли я о написании книги. Вопрос оказался с подвохом, так как она была книгоиздателем. Так родилась серия «How to Break Software». Я написал первую книгу сам до последнего слова и был сильно утомлен процессом, поэтому решил больше не писать в одиночку. Над следующими двумя книгами я работал с соавторами. Хью Томпсон написал книгу «How to Break Software Security», Майк Эндрюс написал «How to Break Web Software», а я помогал им обоим. Это действительно их книги. Я всего лишь написал часть материалов и управлял процессом. Мне нравится писать, но ни Хью, ни Майк не стали бы смущать меня, утверждая, что я пишу лучше, чем они. А вы двое? Я так не думаю. Но факт в том, что без меня никто бы не решился писать. В конечном итоге все, что происходит в моей карьере, заканчивается книгой, а окружающие меня люди становятся соавторами. Будете отрицать?

— Вообще-то читатель сейчас держит в руках доказательство твоих слов. Как тут поспоришь!

Джеймс: Не буду говорить, что я совсем не могу писать в одиночку. «Exploratory Testing» — книга, которая была написана без соавторов. Она появилась из презентаций, которые я проводил на конференциях. Я создал тонну материалов, и со временем их накопилось достаточно на книгу.

А за эту книгу я бы вряд ли взялся, если бы вы двое не согласились помочь. Впрочем, это моя первая книга, которая по-настоящему писалась коллективно. На мой взгляд, каждый из нас внес равный вклад.

— Да, это так, и мы были очень рады возможности поучаствовать. Возможно, в программировании мы сильнее тебя, а Джефф вообще круче всех на этой планете, но ты умеешь создавать легко читаемый текст! Скажи, какая часть книги тебе нравится больше всего?

Джеймс: Мне нравится вся книга. Писать ее было очень интересно. Дело даже не в поиске материала — он у нас уже был. Нам оставалось только записать его. Наверное, если все-таки выбирать любимые части книги, то я выберу интервью. Их было интересно и проводить, и записывать. Надо было с них начинать работу. Беседа с Хун Даном стала для меня настоящим откровением. Он провел для меня экскурсию по своей лаборатории Android, где мы обсуждали философию тестирования, и интервью получилось крайне насыщенным. Я так быстро не записывал со времен аспирантуры. Это было самое ценное время, которое я когда-либо проводил с ним. Я узнал о людях и процессах то, чего не знал раньше. Наверное, это особенность работы журналиста — удается по-настоящему узнать тему.

— А чем бы ты занимался, если бы не попал в тестирование?

Джеймс: В технологической области я, наверное, занимался бы инструментами для разработки и стал бы евангелистом. Мне нравится упрощать создание про-

граммного кода. Не все программисты так хороши, как Джефф Каролло! Не могу поверить, что мы до сих пор пишем приложения вручную. Навыки разработчика, которые я освоил в 80-е годы, до сих пор востребованы. Это какое-то безумие. Весь технологический мир изменился, а мы все еще пишем на C++. Почему разработка не упрощается? Почему написание ужасного, ненадежного кода до сих пор остается нормой? Писать хороший код должно быть проще, чем плохой. Я бы работал над этой проблемой.

Евангелизм — тоже очень важная штука. Я, например, люблю выступать перед людьми, мне нравится рассказывать инженерам про технологии. Я думаю, что если кто-то захочет нанять меня, чтобы я весь рабочий день общался с разработчиками, то это может быть даже круче, чем тестирование. Если найдете что-нибудь такое, дайте знать.

— А если бы пришлось работать за пределами технологической области?

Джеймс: С этим сложнее, потому что пока я не вижу для себя иной специальности — я все еще горю этой. Думаю, я мог бы вести курсы менеджмента. Вы, парни, часто говорите мне, что я неплохой менеджер; когда-нибудь я сяду и подумаю над тем, как мне удастся не лаять. Может, это станет темой моей следующей книги — «Как быть менеджером и не лаять». Также мне хотелось бы поработать на пользу окружающей среде. Мне нравится эта планета, и я думаю, ее стоит держать в порядке.

Да, и я люблю пиво. Я очень люблю пиво. Когда-нибудь я бы мог сыграть Норма из сериала «Cheers». Могу себе представить: я вхожу в бар, все кричат: «Джеймс!», а тот, кто занимает мой табурет, уступает мне место.

Вот это я называю хорошо сделанной работой. Пожалуй, я бы выписал премию Норму из чистого уважения.

Как мы улучшали тестирование в Google

Все тестирование в Google можно описать одной фразой: забота о качестве — это ежедневная обязанность каждого инженера. Если это делается на совесть — уровень качества растет. Новый код становится чище, ранние сборки — устойчивее. Критичность интеграционного тестирования падает, а системное тестирование фокусируется на проблемах пользователей. Проекты и инженеры спасены от рапущего снежного кома багов.

Если ваша компания достигла такого уровня качества, встает вопрос: «А что дальше?».

Google уже столкнулся с этим «далее». Пока мы совершенствовали роль тестирования, мы встретились с немалым количеством побочных проблем. В этой главе мы расскажем о наших ошибках и разберем, как тестирование помогает (или мешает) их исправлять. В конце концов, наше тестирование ушло от централизованного управления, повышение продуктивности разработки перестало управляемым из одной точки и разошлось по командам. Мы думаем, что когда уровень тестирования достигает зрелости, такая перестановка сил неизбежна. Модель, в которой разработка отделена от тестирования, больше не жизнеспособна в Google.

Роковые ошибки в процессе тестирования Google

Понятие «тестирование» часто подменяет понятие «качество». Если спросить разработчика, что он делает для качества продукта, нередко можно услышать: «Я его

тестирую». Но смысл тестирования не в улучшении качества. Качество должно быть встроено в продукт по умолчанию, а не привинчено к нему позже, поэтому качество должен обеспечивать разработчик, и точка. Итак, встречайте роковую ошибку номер один: тестировщики превратились в «костыли» для разработчиков. Чем меньше мы заставляем разработчиков думать о тестировании, чем сильнее его упрощаем для них, тем меньше они им занимаются.

Когда мы сидим на удобном диване и смотрим телевизор, пока кто-то стрижет наш газон, работает та же схема. Мы ведь и сами в состоянии его подстричь. Что еще хуже, пока его кто-то стрижет, мы валяемся на диване нога на ногу и ничего не делаем. Компании по стрижке газонов настолько упростили для нас всю работу, что мы с легким сердцем поручаем ее кому-то другому. Если тестирование выделяется в удобный сервис, о котором разработчики могут не думать, они и не будут думать. Мы сделали процесс тестирования слишком простым, и наши разработчики стали слишком ленивыми.

Проблема усугубляется тем, что тестирование в Google выделено в отдельное направление. Качество — это не только проблема кого-то другого, это проблема другого отдела. Как и в случае с газоном, легко найти ответственного и повесить на него все, что пошло не так.

Роковая ошибка номер два тоже связана с разделением разработчиков и тестировщиков: тестировщики отождествляются со своей ролью, а не с продуктом.

Если фокус не на продукте — продукт всегда страдает. В конце концов, цель всей разработки — создание продукта, а не программирование, тестирование или документирование. Каждый инженер работает на продукт. Его должность при этом вторична. Признак здоровой компании — когда люди говорят «Я работаю в Chrome», а не «Я работаю тестировщиком».

Пару лет назад я видел на конференции по тестированию футболку с надписью «Я тестирую, следовательно, я существую» на греческом и английском языках. Не сомневаюсь, что ее автор дьявольски умен, но этот агрессивный слоган раздувает роль тестировщика, хотя она этого не стоит. Ни одна роль этого не стоит. Все участники команды работают над продуктом, а не над отдельными его частями. Сначала появился продукт, а потом вылупился процесс. Зачем же нам процесс, если не для создания хорошего продукта? Пользователи любят продукты, а не процессы.

Разделение разработки и тестирования в Google подчеркивает эту разницу в ролях и мешает тестировщикам осознавать свое место в продукте.

Роковая ошибка номер три: тестировщики часто ставят артефакты тестирования выше самого продукта. Ценность тестирования — в действии, а не в артефактах.

Каждый артефакт, который создает тестировщик, уже вторичен по отношению к исходному коду. Тест-кейсы и план тестирования менее важны. Даже баг-

репорты менее важны. Действия, которые стоят за этими артефактами, — вот что приносит пользу. К сожалению, восхваляя стопку баг-репортов, представленную тестировщиками во время ежегодного отчета, мы забываем о продукте. Ценность артефактов определяется тем, как они влияют на исходный код, а значит, на продукт.

Выделенная команда тестировщиков часто фокусируется на создании и сопровождении артефактов тестирования. А для продукта лучше всего, если вся деятельность по тестированию направлена только на исходный код. Тестировщики должны ставить продукт во главу угла.

Напоследок, пожалуй, самая показательная роковая ошибка. Вопрос: как часто после выпуска продукта пользователи находили баги, укрывшиеся от самого дотошного процесса тестирования? Ответ: почти всегда. Во всех продуктах, которые мы выпускали, при эксплуатации находились ошибки, которые не нашла команда тестирования. И так не только в Google. Во время написания книги мы все втроем работали в проекте Google+. Можем сказать, что довольно много коварных багов обнаружили внутренние пользователи, то есть другие сотрудники Google, которые просто использовали продукт. Мы прикидывались пользователями, а они были пользователями.

Не важно, кто именно тестирует продукт, главное, что тестирование проводится. Внутренние пользователи, доверенные внешние тестировщики, краудсорсеры, ранние пользователи — все они в более выгодном положении, чем сами тестировщики продукта. По факту, чем меньше инженер по тестированию тестирует сам и чем больше он помогает выполнять эту работу другим, тем лучше для продукта.

О'кей, и что же со всем этим делать? Как взять только хорошее из организации тестирования в Google и переориентировать все процессы на продукты и команды? Мы только вступаем на эту неисследованную территорию, поэтому можем только предполагать. Но нам кажется, что тренды, которые мы описываем в книге, задают тон будущего тестирования в Google и за его пределами. Более того, выделившиеся роли разработчика в тестировании и инженера по тестированию — это изменение в сторону как раз этого будущего.

В Google эти две роли все сильнее расходятся. Разработчик в тестировании все больше напоминает разработчика, а инженер по тестированию идет в противоположную сторону и почти сливаются с пользователем. Это происходит естественно, в результате взросления процессов разработки. С одной стороны, этот тренд объясняется такими техническими новшествами, как сжатые циклы разработки и непрерывная доступность последней сборки для разработчиков, тестировщиков и пользователей. У нас стало больше возможностей подключать неинженерных участников к разработке. С другой стороны, взросление процессов разработки привело к тому, что качество стало общей заботой, а не только обязанностью инженеров, в должности которых есть слово «тестирование».

Будущее разработчика в тестировании

Если коротко, то мы думаем, что у разработчиков в тестировании нет будущего. Все-таки они — разработчики, и точка. Google платит им как разработчикам и оценивает их результаты по тем же критериям, что и разработчиков. Они даже называются одинаково — разработчики. Столько совпадений может говорить только об одном: это одна и та же роль.

Хотя роль, как нам кажется, обречена на вымирание, сама работа никуда не исчезнет. Эта деятельность — часть волшебной формулы успеха Google. Разработчики в тестировании обеспечивают тестируемость, надежность, возможность отладки и многое другое. Если мы будем рассматривать эти свойства как фичи продукта, так же как пользовательский интерфейс и другие функциональные компоненты, то разработчики в тестировании станут разработчиками, ответственными за реализацию этих фич. Мы думаем, что эта эволюция роли скоро произойдет не только в Google, но и в других зрелых IT-компаниях. Есть ли более эффективный способ поднять статус тест-разработки, чем рассматривать ее наравне с остальными фичами продукта?

Сейчас эта часть процесса ущербна. Все пользовательские фичи официально управляются менеджером продукта, а создаются разработчиками. Код для этих фич управляется, и сопровождается, и поддерживается четкими автоматизированными процессами. При этом код тестов пишут разработчики в тестировании, а фактически управляют им инженеры по тестированию. Как так вышло? Это пережиток исторически сложившегося распределения ролей. Но эволюция не дремлет, и уже пора работать с кодом тестов так же, как и с кодом продукта: его должны писать разработчики, а управлять им должны менеджеры продукта.

Следующий вопрос: из каких разработчиков получатся хорошие разработчики тестов? Кому из них можно доверить ответственность за качество? Кто из них серьезно относится к этой работе? Что, если поступить наоборот? В Google, где роль разработчиков в тестировании давно устоялась, можно просто взять и одним махом превратить их всех в разработчиков. И тема закрыта. Но мы считаем, что это топорное решение. Каждый разработчик может выиграть от экскурсии в мир ответственности за фичи качества¹. Тем не менее принуждать людей к этому несерьезно, да это и не в духе Google. Вместо этого ответственность за фичи тестирования должна отдаваться новым участникам команд, особенно менее опытным.

¹ В Google есть программа подготовки инженеров по эксплуатационной надежности (SRE, Service Reliability Engineer), которая называется Mission Control. Разработчик, прошедший шестимесячную программу в роли SRE, получал приличную премию и кожаную куртку с эмблемой Google Mission Control.

Вот вам обоснование. Фичи тестирования — это срез всего продукта. Поэтому разработчики, которые их создают, изучают продукт сверху донизу, от пользовательских интерфейсов до API. Разве можно еще глубже погрузиться в продукт и изучить его строение и архитектуру? Ответственность за тестовую фичу, будь то построение ее с нуля, изменение или сопровождение, — прекрасный старт для любого разработчика в команде. Мы бы даже сказали — лучший старт. С приходом новых участников тест-разработчики освобождают им место и переходят на разработку функциональности. Ребята не застаиваются, и со временем все разработчики становятся подкованными в тестировании и уже серьезно относятся к качеству.

Для младших разработчиков и выпускников колледжа разработка тестов — самый удачный старт. Во-первых, они знакомятся с проектом в целом, во-вторых, большая часть тестового кода не участвует в выпуске, а значит, они далеки от давления и возможного стыда за баги, которые найдут пользователи. По крайней мере, на этом этапе карьеры.

Главное отличие этой модели с единой ролью в том, что экспертиза тестирования более равномерно распределена среди разработчиков. Сейчас же в Google, наоборот, все сконцентрировано у разработчиков в тестировании. Одно это важное изменение увеличило бы производительность, избавив нас от бутылочного горлышка в виде разработчиков в тестировании. Кроме того, отсутствие различий в должностях инженеров привело бы к свободным переходам от разработки тестов к разработке фич: один продукт, одна команда, одна роль.

Куда движется роль инженера по тестированию

Потребность в хороших тестировщиках как никогда высока. И все же мы думаем, что ее пик уже прошел. Традиционно тестировщики писали и выполняли тест-кейсы, проводили регрессионное тестирование. Но сейчас большая часть этой работы стала доступна в новых форматах, более полных и дешевых.

Большая часть новых возможностей появилась благодаря технологическому совершенствованию механизмов поставки программных продуктов. Во времена еженедельных или даже ежемесячных сборок и кропотливых интеграций важно было иметь под рукой тестировщиков, которые умели искать баги, имитируя поведение пользователей. Разумеется, баги нужно было найти до того, как продукт попадет в руки миллионов людей, когда трудно будет отслеживать дефекты и накатывать обновления. Эти времена давно прошли. Распространение приложений через веб позволяет выпускать продукт для ограниченного числа пользователей, получать от них обратную связь и быстро обновлять продукт. Препятствий для

общения разработчиков с пользователями больше нет. Срок жизни багов уменьшился с нескольких месяцев до нескольких минут. Механизмы распространения и получения обратной связи настолько хороши, что мы пишем, выпускаем, исправляем и снова выпускаем продукт быстрее, чем большая часть аудитории успевает заметить дефект. Мы можем дать продукт только группе доверенных тестировщиков или внутренних пользователей, найти ранних последователей или выкатить для всей аудитории. Какое место в этой схеме занимает команда профессиональных инженеров по тестированию? Пользователи гораздо лучше защищены от проблем, чем раньше, поэтому настало время перераспределить наши ресурсы тестирования.

Подведем итог: кому бы вы доверили тестирование своего программного продукта? Высокооплачиваемым, нанятым экспертам по исследовательскому тестированию, которые всеми силами пытаются предугадать реальные сценарии использования, надеясь найти критические баги, или огромной армии настоящих пользователей, которые с горящими глазами ищут баги и сообщают о них? Общаться с реальными пользователями, готовыми искать баги в обмен на ранний доступ к продукту, еще никогда не было так просто. А с ежедневными или даже ежечасными обновлениями реальные риски для самих пользователей минимальны. В этом новом мире роль инженера по тестированию должна быть переосмыслена.

Мы думаем, что инженеры по тестированию станут проектировщиками тестов. Небольшое количество тест-дизайнеров будет оперативно планировать покрытие тестами, составлять тепловую карту рисков и туры для приложения (помните третью главу?). Получив обратную связь от доверенных тестировщиков, внутренних и ранних пользователей или краудсорсеров, тест-дизайнер оценит покрытие, пересчитает риски, убедится в их снижении и перераспределит активности. Тест-дизайнеры будут определять, где требуется специальная экспертиза по безопасности, конфиденциальности данных и производительности, а где — исследовательское тестирование, и передавать эту работу внешним специалистам. Их задачей будет создать или настроить инструменты для сбора и анализа всех входных данных. Им не придется ни создавать, ни выполнять тесты. Им не нужно будет *тестировать*. О'кей, может быть, «не нужно» — это слишком громкая фраза, но тестирования в работе тест-дизайнеров должно быть очень мало. Их работой будет проектирование, организация и управление ресурсами тестирования, которые по большей части бесплатны.

Мы думаем, что роль инженера по тестированию будет постепенно смещаться в сторону узкой специализации, например безопасности, или в сторону управления процессами тестирования, выполняемого другими. Это важная и серьезная должность, которая требует большого опыта. Возможно, она даже будет оплачиваться значительно выше, чем сейчас, но и людей на нее будет нужно намного меньше.

Что станет с тест-директором и тест-менеджером

Как все эти ролевые изменения отразятся на менеджерах, директорах и вице-президентах по тестированию? Их станет меньше. Те из них, у кого есть технические знания, перейдут на другие роли, более подходящие их инженерной специфике. Другие останутся идеальными лидерами и будут координировать работу инженеров по тестированию и разработчиков (мы помним, что они теперь ориентированы на качество). Но они не будут управлять конкретным проектом и отвечать за его качество. За тестирование должны отвечать люди, непосредственно работающие с продуктами, а не централизованная организация, слабо связанная с продуктами и процессами разработки.

Будущее инфраструктуры тестирования

Инфраструктура тестирования в Google все еще клиентская. Множество тестов Selenium и WebDriver, написанных на Java или Python, хранится в репозитории, собирается и разворачивается на выделенных виртуальных машинах с помощью shell-скриптов. Тест-драйверы внедряют тестовую логику на Java в браузер, сами при этом выполняются как приложение в ОС. Решение работает, но инфраструктуру пора перестраивать, потому что этот метод требует дорогостоящих человеческих и машинных ресурсов для создания и выполнения тестов.

Инфраструктура тестирования постепенно перемещается в облако. Репозитории тест-кейсов, редакторы кода тестов, средства записи и выполнения — все они будут работать прямо на сайте или в расширении браузера. Подготовка, прогон и отладка тестов эффективнее, когда они выполняются на одном языке и в одном контексте с приложением. В наше время чаще всего речь идет о веб-приложениях, и так не только в Google. В нативных проектах, не связанных с вебом (например, платформенные приложения Android и iOS), тестирование будет управляться из веба и использовать адаптированные веб-фреймворки тестирования. Native Driver¹ — прекрасный пример перехода на модель «веб — первичен, нативность — вторична».

В современной быстро меняющейся среде, где проекты вместе со своими тестами возникают и исчезают все быстрее, будет оставаться все меньше внутренних, за-

¹ Пост в тестовом блоге Google по поводу Native Driver: <http://googleopensource.blogspot.com/2011/06/introducing-native-driver.html>

точенных под один проект тестовых фреймворков и выделенных машин для тестов. Разработчики тестов начинают вкладывать больше усилий в развитие опенсорс-проектов, объединять и выполнять их в облачных мощностях. Selenium и WebDriver задают модель разработки инфраструктуры, сопровождающейся сообществом и финансируемой корпорациями. Будет возникать еще больше таких проектов, повысится степень интеграции между открытыми тестовыми фреймворками, открытыми багтрекинговыми системами и системами управления исходным кодом.

Общий доступ к тестовым данным, тест-кейсам и инфраструктуре компенсирует отход от секретности и эксклюзивности. Закрытые проприетарные тестовые инфраструктуры на самом деле означают всего лишь дорогостоящие, медленно работающие продукты, которые, скорее всего, нельзя повторно использовать даже в рамках одной компании. Тестирующие будущего будут стараться использовать совместно как можно больше кода, тестов и информации о багах, полученных от сообщества. Появятся новые формы краудсорс-тестирования и создания тестов, и добрая воля пользователей перевесит надуманные преимущества от приватности всего.

Более открытый, облачный подход к тестированию сэкономит компаниям средства, даст разработчикам тестовых инфраструктур больше прозрачности и, что самое важное, позволит разработчикам тестов сосредоточиться на тестовом покрытии своих проектов, а не на инфраструктуре. Циклы выпуска ускорятся, и качество продуктов вырастет.

В завершение

Тестируанию, которое мы знаем и любим, приходит конец. Кого-то эта новость огорчит. Тем, чья карьера зависит от текущей схемы работы, придется намного сложнее. Так или иначе, разработка программного обеспечения фундаментально изменилась с приходом гибких методов, непрерывной сборки, раннего вовлечения пользователей, краудсорсинг-тестирования и онлайн-распространения приложений. Держаться за стандарты десятилетней давности — значит обрекать себя на отставание.

Этот переход уже происходит в Google, хотя и не все еще в курсе. Инженеры, менеджеры и директора централизованного тестирования рассеиваются по командам конкретных проектов. Переход заставляет их действовать быстрее, уделять меньше внимания процессу тестирования и больше — самому продукту. Мы в Google заметили этот переход чуть раньше, чем другие компании. Мир скоро изменится для всех тестирующих. Примите эти изменения и управляйте ими, чтобы не потерять свою релевантность как тестирующих.

Тест-план для Chrome OS

Статус: черновик

Контакт: chromeos-te@google.com

Автор: jarbon@google.com

Обзор тем

- **Риски.** В Chrome OS много областей для тестирования: браузер, интерфейс взаимодействия с пользователем, прошивка, оборудование, поддержка сети, синхронизация данных пользователя, автоматические обновления и даже специфическое оборудование, созданное внешними производителями. Чтобы справиться с этими задачами, разработаем стратегию тестирования, основанную на рисках. Команда начнет тестирование с самой рискованной области, а потом будет последовательно спускаться вниз по списку, к менее рискованным областям. Мы будем полагаться на тщательное юнит-тестирование и качественный код команды разработки, которые составляют фундамент качества всего продукта.
- **Автоматизация тестирования оборудования.** Учитывая разнообразие оборудования и непрерывную сборку ОС, важно прогонять тесты для каждой сборки по всей матрице оборудования. Так мы быстро выявим регрессии и сможем диагностировать место возникновения проблемы, будь то код приложения, железо или конфигурация среды. Например, тест может падать только на оборудовании HP в браузере сборки X при определенных настройках беспроводной связи.

- **Поддержка быстрых итераций.** У Chrome OS жесткий график выпуска, поэтому важно как можно раньше выявлять баги и определять точные условия их воспроизведения. Все тесты будут сначала запускаться на машине разработчика, чтобы баги не попадали в основную ветку кода. Большой набор автотестов ускорит выявление регрессионных багов.
- **Открытые инструменты и тесты.** Chromium OS — платформа с открытым кодом. К тому же есть определенные требования от производителей оборудования. Поэтому команда тестирования должна убедиться, что все инструменты и тесты могут быть открыты наружу и использоваться за пределами Google.
- **Chrome OS — основная платформа браузера.** Команда тестирования браузера Chrome перейдет к использованию Chrome OS как основной платформы. Обеспечение тестируемости кода, автоматизация тестов и остальные активности в первую очередь ориентируются на Chrome OS и только после — на другие платформы. Роль браузера Chrome станет решающей, ведь это единственный пользовательский интерфейс Chrome OS. Железо служит поддержкой браузеру и полностью работает на него. Для Chrome OS планка качества браузера Chrome должна быть очень высокой.
- **Предоставление данных.** Обеспечение качества — это не цель команды тестирования. Качество — забота всех участников проекта, даже внешних производителей оборудования и участников опенсорс-сообщества. Цель тестировщиков — снизить риски, найти как можно больше багов и проблем, предоставить команде метрики и общую оценку рисков. Тестирование, разработка, менеджмент и все заинтересованные стороны снаружи Google имеют право голоса и влияние на качество Chrome OS.
- **Тестируемость и сотрудничество.** Тестируемость всегда была камнем преткновения, особенно для команд Google apps, внешних разработчиков, да и внутри компании. Тестировщики скооперируются с командами Android и WebDriver, чтобы повысить тестируемость и полноценно поддерживать браузер Chrome в Chrome OS. Это увеличит производительность автоматизации тестирования в командах Google apps. Так Chrome будет постепенно становиться идеальной платформой для тестирования любых приложений со сторонними веб-страницами.

Анализ рисков

Команда тестирования выполнит анализ рисков, чтобы:

- убедиться, что вся команда понимает риски качества продукта;

- убедиться в том, что команда тестирования в каждый момент сфокусирована на задаче, которая принесет максимум пользы;
- выработать набор правил для оценки новых рисков на основании новых данных, которые будут поступать в ходе развития продукта.

В процессе анализа рисков мы составим список всех известных фич и возможностей продукта. Команда тестирования оценит абсолютный риск каждой области. Этот риск рассчитывается из вероятности и частоты появления сбоя в сочетании с серьезностью его последствий для пользователя и для бизнеса. «Смягчающие обстоятельства» (тест-кейсы, автоматизация, тестирование внутренними пользователями, тестирование на стороне производителя оборудования и т. д.) могут снизить значение риска. Все компоненты отранжируются по риску и в соответствующем порядке отправятся на разработку тест-кейсов, автоматизацию и т. д.

В итоге мы должны знать, какие риски есть в продукте и какие из доступных ресурсов мы можем использовать для их методичного сокращения.

Непрерывное тестирование каждой сборки

Кроме юнит-тестов с использованием Buildbots, для каждой непрерывной сборки должны выполняться следующие тесты:

- смок-тесты (автоматизация фич и багов с приоритетом P0);
- тесты производительности.

Ежедневное тестирование лучших сборок

Лучшая сборка дня будет проходить через следующее тестирование:

- ручная проверка набора функциональных приемочных тестов (можно ограничиться одним видом оборудования в день);
- автоматизированное функциональное регрессионное тестирование;
- система непрерывного тестирования популярных веб-приложений подхватывает лучшую сборку и прогоняет на ней свои тесты (не только автоматизированные, здесь может проводиться и ручное тестирование).

Тесты на стабильность, нагрузочные и продолжительные тесты проводятся сразу после сборки версии. Тесты прогоняются непрерывно на ежедневных сборках, пока не перестают выявлять баги, и переходят на еженедельное выполнение.

Непрерывно проводится ручное исследовательское тестирование и тестовые туры.

Автоматизируются тесты AutoUpdate.

Тестирование перед выпуском

Тестируется каждый «релиз-кандидат» сборки, для всех каналов.

- **Совместимость с сайтами.** Команда тестирования браузера Chrome проверяет 100 самых популярных сайтов на Chrome OS.
- **Тестирование сценариев.** Проверка всех демосценариев Chrome OS, которые часто показывают партнерам и на конференциях (может быть ограничено максимум двумя-тремя сценариями).
- **Проверка багов с приоритетом P0.** Проверка всех исправленных багов P0. Проверка 80% багов P1, заведенных с момента последнего выпуска.
- **Полная серия тестов на нагрузку и стабильность.** Проведение нагрузочного тестирования и тестирования стабильности.
- **Ручные тест-кейсы Chrome OS.** Выполняются все ручные тест-кейсы Chrome OS. Можно распределить их среди тестировщиков с разным оборудованием.

Ручное и автоматизированное тестирование

Ручное тестирование необходимо. Оно особенно важно на ранней стадии проекта, когда пользовательский интерфейс и другие фичи уже быстро развиваются, а работа по улучшению тестируемости и автоматизации только началась. Chrome OS делает ставку на простоту и интуитивную понятность своего интерфейса, поэтому ручное тестирование проводить обязательно. Машины до сих пор не умеют это тестировать.

Автоматизация — залог долгосрочного успеха и результативности тестовой команды и защита от регрессионных багов. Так как автоматизация встроена в браузер, большая часть ручных тестов (да здравствует продуктивность!) тоже автоматизируется.

Разработка и качество тестов

Команда разработчиков больше других групп по количеству людей и обладает гораздо большим знанием о компонентах и технических подробностях списков изменений. Мы хотим, чтобы разработчики обеспечивали полный набор модульных и узконаправленных системных тестов с помощью Autotest.

Команда тестирования фокусируется больше на сквозных и интеграционных тестовых сценариях. Она возьмет на себя функциональность, заметную пользователю. Тестировщики занимаются проверкой взаимодействия компонентов, стабильностью, крупномасштабным тестированием и отчетностью.

Каналы выпуска

Мы должны использовать тот же подход, который команда браузера Chrome успешно применяла в своем проекте: разделить пользователей на группы по их терпимости к ошибкам и готовности предоставить обратную связь и запустить для них разные каналы. В итоге мы получим ряд каналов с разными уровнями уверенности в качестве. Работа с аудиторией будет вестись импульсно, по необходимости, а не постоянно, что позволит нам уменьшить наши затраты. Мы сможем экспериментировать с реальным использованием продукта до крупномасштабного запуска и снизим риск всего продукта.

Обратная связь

Отзывы пользователей очень важны для проекта. Нужно вложиться в то, чтобы им было предельно просто отправить нам обратную связь. И не забыть о том, что нам нужно будет обрабатывать данные.

- **Расширение GoogleFeedback.** Чтобы отправить сообщение, пользователи могут легко выделить и отправить ошибку для любого URL-адреса. Расширение группирует полученные от пользователей данные и выводит их на информационную панель для анализа. Команда тестирования приложит усилия, чтобы GoogleFeedback был интегрирован в Chrome OS, и расширит его отчетность на ChromeUX.
- **Панель информации о багах.** У приближенных к разработке пользователей проекта должны быть простые инструменты, чтобы заводить новые или про-

смотреть существующие баги прямо в браузере Chrome OS. В перспективе идея должна развиться в общую систему отображения информации в режиме наложения, чтобы инженер сразу видел состояние проекта и его качество.

Команда тестирования будет настойчиво продвигать настройку такой системы, в том числе и для нестандартных конфигураций.

Репозитории тест-кейсов

- **Ручные:** все ручные тест-кейсы хранятся в TestScribe. Идет работа над созданием репозитория тест-кейсов для code.google.com.
- **Автоматизированные:** все автоматизированные тест-кейсы находятся в хранилище кода в формате Autotest. Все тесты версионизированы, доступны и располагаются рядом с тестируемым кодом.

Панели мониторинга тестов

Нужно будет быстро обрабатывать и распространять большой объем данных, поэтому команда тестирования возьмет на себя создание специальных информационных панелей для метрик качества. Это позволит командам быстро получать высокоуровневые данные: индикаторы качества (зеленые/красные сигналы о прохождениях или падениях тестов), общие результаты ручного и автоматизированного тестирования. Если нужно — можно будет копнуть глубже и получить детальную информацию о багах.

Виртуализация

Очень важно, особенно на ранней стадии, вложиться в виртуализацию образов Chrome OS. Это уменьшит зависимость от физического оборудования, поможет проводить регрессионное тестирование с помощью Selenium и WebDriver на целых фермах серверов и облегчит поддержку тестирования и разработки Chrome OS прямо на рабочих машинах инженеров.

Производительность

В двух словах производительность — ключевая фича Chrome OS. Поэтому она выделена в отдельный проект в команде разработки. Команда тестирования старается помочь рассчитать, предоставить и проанализировать показатели продуктивности в лаборатории, но не занимается непосредственно тестированием производительности.

Нагрузочное тестирование, продолжительное тестирование и тестирование стабильности

Команда тестирования создает и выполняет продолжительные тесты на физическом оборудовании в лаборатории. Не забыть про внедрение неисправностей (fault injection).

Фреймворк выполнения тестов Autotest

Команды тестирования и разработки решили использовать Autotest как основной фреймворк для автоматизации тестов. Autotest удачно прошел проверку в сообществе Linux, использовался в нескольких внутренних проектах, и, кроме того, он распространяется с открытым кодом. Autotest поддерживает локальное и распределенное выполнение. Этот фреймворк умеет подключать и другие оснастки функционального тестирования (например, WebDriver), так что у нас будет единый интерфейс для выполнения, распределения тестов и отчетности.

Хотим отметить, что основная команда разработки инструментов тестирования работает над поддержкой Autotest в Windows и Mac.

Производители железа

Производители железа играют важную роль в проверке качества сборок Chrome OS. Команды тестирования и разработки создают релевантные ручные и автома-

тизированные тест-кейсы, при помощи которых производители проверяют сборки и оборудование на своей стороне. Кроме них команда тестирования сотрудничает с самыми популярными производителями и включает разные варианты железа в повседневное тестирование. Так мы сможем как можно раньше выявлять любые проблемы или регрессии, связанные с ним.

Лаборатория проверки оборудования

В лаборатории собрано много нетбуков и других устройств, на которых можно регулировать сетевые настройки (для проводного и беспроводного соединений), управлять питанием и т. д. Это инфраструктура для тестирования отдельных сервисов (например, беспроводной связи). Лабораторные машины в основном управляются инфраструктурой HIVE.

Фермы для сквозных автотестов

Команда тестирования создаст ферму нетбуков для прогона тестов и настроит передачу данных о состоянии матрицы аппаратных и программных средств. Фермы распределяются по офисам в Маунтин-Вью, Киркленде и Хайдарабаде для возможности локального доступа к лаборатории и организации практически круглосуточного выполнения тестов и отладки.

Тестирование AppManager в браузере

Основной браузер Chrome OS — это Linux-версия Chrome с пользовательским интерфейсом и фичами, специализированными для Chrome OS. Большая часть механизмов отображения страниц и функциональности такая же, однако есть значительные различия между базовым браузером и его разновидностью в Chrome OS. Например, закрепляемые вкладки, менеджер загрузки, запуск приложений, платформенные элементы управления, беспроводная связь и т. д.

- Chrome OS — главная платформа для основного тестирования браузера Chrome (ручного и автоматизированного).
- Команда разработки браузера определяет сборку, которую нужно интегрировать, на основании качества сборки и текущих фич Chrome OS.

- Для каждой «релиз-кандидат» версии OS команда браузера проводит стандартный набор тестов совместимости сайтов/приложений (топ-300 сайтов) с Chrome OS.
- Тесты совместимости сайтов/приложений частично автоматизируются с помощью WebDriver и интегрируются в автозапуски при сборках или в обычные лабораторные запуски, чтобы рано получать информацию о серьезных регрессиях, специфических для Chrome OS.
- Команда внешних тестировщиков создает и выполняет пакет ручных тестов для фич Chrome OS, связанных с браузером и менеджером приложений.
- Как только API будет реализован, внешние тестировщики автоматизируют пакет ручных тестов Chrome OS.
- Chrome OS Chromebot должен иметь версии для Linux и Chrome OS и запускаться на всей функциональности Chrome OS, а не только веб-приложений.
- Ручное исследовательское тестирование и туры помогут выявить пользовательские проблемы, связанные с функциональностью, удобством, простотой использования и т. д.

Тестируемость браузера

Именно через браузер пользователь оперирует основными элементами интерфейса и использует фичи Chrome OS. Большая часть компонентов BrowserUX либо не подходит для тестирования, либо может тестироваться только через низкоуровневые интерфейсы IPC AutomationProxy за пределами браузера. Для Chrome OS мы унифицируем тестирование веб-приложений, пользовательского интерфейса Chrome и функциональности. Мы стараемся избавиться от низкоуровневых системных тестов. Мы хотим, чтобы Chrome стал самым удобным браузером для тестирования веб-приложений, чтобы внешние команды веб-разработки начинали тестирование своих проектов именно с него. Вот что мы делаем для этого:

- **Портование Selenium и WebDriver в Chrome OS.** Это основные тестовые фреймворки для современных веб-приложений. Команды браузера Chrome и Chrome OS, скорее всего, возьмут на себя специфические для Chrome аспекты WebDriver. Это станет шагом к созданию надежного, удобно тестируемого интерфейса для разработчиков приложений и внешних тестировщиков.
- **Доступ к пользовательскому интерфейсу и функциональности Chrome через JavaScript DOM.** Это позволит тестам WebDriver работать с пользовательским интерфейсом и функциональными аспектами Chrome. Функцио-

нальность доступна через те же методы, что и отключение, и спящий режим, и через которые с ChromeViews работают специалисты по доступности для людей с ограниченными возможностями (например, raman@).

- **Высокоуровневые сценарии.** Сотрудничество с командой WebDriver для расширения основного API WebDriver сначала в «чистый» JavaScript, а потом в высокоуровневые скрипты записи и воспроизведения с параметрами (например, «Google Search: <критерий>»). Так мы ускорим внутреннюю и внешнюю разработку тестов, которые, работая с WebDriver, все еще требуют большой работы по поиску элементов и сложны в поддержке из-за быстрого изменения пользовательского интерфейса.

Оборудование

Chrome OS должен соответствовать требованиям многих производителей оборудования. Нужно тестировать на основных платформах производителей железа, чтобы обеспечить интеграцию между физическим оборудованием и Chrome OS. Мы создаем тесты, которые проверяют:

- **управление питанием:** циклы питания от сети и батареи, сбои, управление питанием аппаратных компонентов и т. д.;
- **сбои оборудования:** что Chrome OS сможет обнаружить и как произведет восстановление?

График

4-й квартал 2009 года:

- Определить ручные приемочные тесты, выполнять их для непрерывных сборок.
- Определить основные тесты проверки качества и выполнять их для каждого крупного релиза.
- Запустить аппаратную лабораторию.
- Провести анализ рисков.
- Автоматизировать выполнение сквозных сценариев для непрерывных сборок на нетбуках в лаборатории.

- Настроить поддержку Hive для визуализации данных на виртуальных и физических машинах.
- Портировать WebDriver и Selenium на Chrome OS.
- Автоматизировать некоторые тесты для популярных веб-приложений.
- Выбрать тестовую оснастку для разработчиков и тестирующих.
- Запустить интеграцию Google Feedback в Chrome OS.
- Сформировать основную команду тестирования: люди и процессы.
- Автоматизировать тесты для аудио и видео.
- Полностью спланировать тестирование с учетом рисков.
- Спланировать ручное тестирование пользовательского интерфейса.

1-й квартал 2010 года:

- Настроить панели мониторинга данных о качестве.
- Автоматизировать тесты автообновления.
- Настроить поддержку автоматизированного тестирования производительности в лаборатории.
- Настроить лаборатории в Хайдарабаде, Киркленде и Маунтин-Вью, к этому времени в них должно уже проводиться тестирование.
- Chromebot для Linux и Chrome OS.
- Обеспечить поддержку тестируемости для основных функций и пользовательского интерфейса Chrome OS.
- Создать набор функциональных регрессионных автоматизированных тестов для Chrome OS.
- Включить Chrome OS в регрессионные тесты веб-приложений для ферм Selenium.
- Создать прототип поддержки записи и воспроизведения для браузера и тестирования пользовательского интерфейса.
- Автоматизировать тест-кейсы ChromeSync E2E.
- Настроить тестирование стабильности и внедрения сбоев.
- Настроить тестирование сетевых компонентов.
- Проводить регулярное исследовательское ручное тестирование и туры (чтобы задобрить Джеймса).

2-й квартал 2010 года:

- Риски уже должны быть снижены из-за тестирования и автоматизации.

3-й квартал 2010 года:

- Риски должны быть снижены еще больше из-за тестирования и автоматизации.

4-й квартал 2010 года:

- Все риски минимизированы, работа автоматизирована, новых проблем нет, пользовательский интерфейс и функциональность остаются неизменными. Команда тестирования завершает работу.

Ключевые моменты тестирования

- Ведущий инженер по тестированию для платформы Chrome OS.
- Ведущий инженер по тестированию для браузера Chrome.
- Ведущий инженер автоматизации браузера.
- Панель мониторинга статусов тестирования и метрик.
- Определение и выполнение ручных приемочных тестов.
- Определение и выполнение ручных регрессионных тестов командами внешних тестировщиков.
- Совместимость приложений, базовый пользовательский интерфейс и функциональность браузера (ручные тесты).
- Аудио и видео.
- Стабильность и внедрение сбоев.
- Тестирование доступности для пользователей с ограниченными возможностями.
- Аппаратные лаборатории: Киркленд, Хайдарабад и Маунтин-Вью.
- Автоматизация ферм.
- Приемка оборудования.
- Общее руководство и финансирование.

Необходимые документы и ресурсы

- Анализ рисков.
- Аппаратные лаборатории.
- Автоматизация ферм E2E.
- Инфраструктура управления виртуальными и физическими машинами.
- Инструменты для отображения служебной информации.
- Ручные приемочные тесты.
- Информационная панель с результатами тестирования.
- Тесты проверки железа для производителей.
- Сводная панель с информацией об использовании и работоспособности оборудования.
- План ручного функционального тестирования Chrome OS.

Тестовые туры для Chrome

Туры тестов:

- Тур покупателя
- Тур студента
- Тур международных звонков
- Тур ориентиров
- Тур «не спим всю ночь»
- Тур предпринимателя
- Тур неблагополучных районов
- Тур персонализации

Тур покупателя

Описание: шопинг — любимое времяпровождение для многих, особенно во время поездок в другие страны, там всегда можно купить что-то новенькое. В некоторых городах сами магазины стали главными достопримечательностями. В Гонконге, например, расположен один из самых больших торговых центров мира, под крышей которого расположено более 800 магазинов.

Коммерция не чужда и области разработки ПО. Хотя и не в каждом приложении можно потратить деньги, но во многих из них есть возможность что-нибудь купить. Движение в ту сторону стало особенно заметным сейчас, когда дополнительно загружаемый контент стал чем-то привычным. Тур покупателя предлагает пользователю совершить покупки везде, где только можно в тестируемом приложении. Так мы проверяем, может ли пользователь покупать без проблем и не тратить на это много времени.

Применение: Chrome открывает перед пользователем дверь в мировой торговый центр, где возможности потратить деньги безграничны. Конечно, протестировать работу каждого интернет-магазина невозможно. Мы можем только проверить доступность и работоспособность большинства из них. Вот список крупнейших интернет-магазинов (по количеству трафика на их сайтах):

- eBay (www.eBay.com)
- Amazon (www.amazon.com)
- Sears (www.sears.com)
- Staples (www.staples.com)
- OfficeMax (www.officemax.com)
- Macy's (www.macys.com)
- NewEgg (www.newegg.com)
- Best Buy (www.bestbuy.com)

Тур студента

Описание: многие студенты уезжают учиться за границу. Обосновавшись в новом городе, они используют местные ресурсы для того, чтобы узнать больше об интересующей их области. Для них есть туры, показывающие им полезные места — библиотеки, архивы, музеи и т. д.

Точно так же в интернете люди пробуют новые технологии, чтобы изучать и узнавать больше по конкретной нужной им теме. Этот тур направлен как раз на это: он помогает воспользоваться приложением и протестировать все его возможности для сбора и организации информации.

Применение: протестировать, насколько хорошо Chrome умеет собирать и систематизировать информацию из разных источников. Например, может ли пользователь получить информацию с нескольких сайтов и объединить ее в обложном документе? Можно ли загрузить и использовать контент в офлайне?

Рекомендуемые области для тестирования

В тур студента для Chrome входят:

- **«Копировать» и «Вставить»:** возможна ли передача разных типов данных через буфер обмена?
- **Перемещение офлайнового контента в облако:** веб-страницы, изображения, текст и прочее.
- **Производительность:** возможность одновременного открытия нескольких документов в разных окнах.
- **Перемещение данных:** перемещение данных между вкладками и окнами, а также между окнами с разными настройками (в обычном режиме и инкогнито).

Тур международных звонков

Описание: во время поездки звонок домой может стать целым приключением. В другой стране легко запутаться в операторах международной связи, карточках оплаты и разнице валют.

При работе с программным продуктом пользователи могут захотеть использовать привычные фичи, но с других платформ, с другими настройками и уровнями доступа. Этот тур подтверждает, что у пользователя не возникнет проблем с использованием продукта, где бы он ни находился.

Применение: просмотр типичных сайтов и использование стандартных возможностей в Chrome на других plataформах (Windows, Mac и Linux) с разными настройками подключения в ОС.

Рекомендуемые области для тестирования

В тур международных звонков для Chrome входят:

- **Операционные системы:** Windows, Mac и Linux.
- **Уровни доступа:** высокие и низкие уровни привилегий.
- **Языки:** сложные языки, системы письменности справа налево.
- **Сетевые настройки:** прокси-сервер, Wi-Fi, проводное подключение, брандмауэр.

Тур ориентиров

Описание: Все очень просто. Используйте компас, чтобы установить ориентир (дерево, камень, склон горы) в нужном вам направлении движения. Доберитесь до него, найдите следующий ориентир и т. д. Если ориентиры идут в одном направлении, вы сможете пробраться даже через дремучий лес. Тур ориентиров для исследовательских тестировщиков напоминает наш выбор ориентиров и движение по ним, чтобы не заблудиться.

Применение: для Chrome этот тур проверяет, легко ли пользователь перемещается от одного объекта к другому. Убедитесь, что пользователи смогут добраться до нужных ориентиров (например, другие окна браузера, открываемые вложения, настройки).

Рекомендуемые ориентиры для Chrome

В тур ориентиров для Chrome входят:

- **Окно браузера:** основное окно браузера для просмотра веб-сайтов.
- **Окно режима инкогнито:** используется для просмотра в режиме повышенной секретности; в левом верхнем углу появляется узнаваемое изображение «шпиона» для оповещения пользователя.
- **Компактная навигационная панель:** это окно браузера доступно из меню; в строке заголовка окна размещается поле поиска.
- **Менеджер загрузки:** вывод списка контента, загруженного пользователем.
- **Менеджер закладок:** менеджер закладок — полноценное окно, в котором отображаются закладки пользователя, которые можно нажать для перехода на страницу.
- **Инструменты разработчика:** менеджер задач, консоль JavaScript и прочее.
- **Настройки:** вызываются из меню в правом верхнем углу.
- **Темы:** страница, на которой пользователи могут выбрать свое оформление Chrome OS.

Тур «не спим всю ночь»

Описание: как далеко вы можете зайти? Тур «не спим всю ночь» предлагает туристам проверить свою выносливость, перемещаясь с одной вечеринки на другую

без перерывов. «Этот тур проверит, на что вы способны. Выдержите? Хватит вас на всю ночь?»

Тур испытывает тестируемый продукт на прочность, чтобы узнать, выдержит ли он активное и продолжительное использование. Ключевой момент — ничего не закрывать, а продолжать один долгий сеанс взаимодействия с приложением. В ходе тура можно найти баги, появляющиеся только после долгого использования приложения.

Применение: открыть много вкладок Chrome, устанавливать расширения, менять темы и продолжать работу в одном сеансе как можно дольше. Страйтесь не закрывать вкладки или окна, когда вы закончите работу с ними, просто открывайте новую страницу. Если тур занимает несколько дней, Chrome нужно оставить открытым на ночь, чтобы продолжить работу на следующий день.

Рекомендуемые области для тестирования

В тур «не спим всю ночь» для Chrome входят:

- **Вкладки и окна:** нужно открыть много вкладок и окон.
- **Расширения:** установить много расширений и запустить их в работу.
- **Продолжительность работы:** оставьте всё открытым надолго.

Тур предпринимателя

Описание: одни путешествуют для развлечения, другие — по делам бизнеса. Этот тур выяснит, насколько легко заниматься бизнесом в стране визита. Есть ли подходящие местные поставщики? С какими бюрократическими тонкостями связано начало работы?

В программных продуктах этот тур проверяет, легко ли пользователь может разрабатывать контент, используя тестируемый продукт. Пользователь проверяет, сколько полезных инструментов дает программный продукт, насколько легко выполняется импорт и экспорт.

Применение: в Chrome есть обширный инструментарий для разработчиков на JavaScript и веб-разработчиков, которые запускают и тестируют свои веб-приложения. Тур нужен для проверки инструментария, генерации примеров скриптов и тестирования онлайн-контента.

Инструменты в Chrome

В тур предпринимателя для Chrome входят:

- **Инструменты разработчика:** просмотр элементов страницы, ресурсов и сценариев, включение отслеживания ресурсов.
- **Консоль JavaScript:** правильно ли выполняется код JavaScript?
- **Просмотр исходного кода:** легко ли читается код с цветовым (или другим) выделением, легко ли перейти к нужному разделу?
- **Менеджер задач:** правильно ли отображаются процессы, легко ли определить, сколько ресурсов потребляет веб-страница?

Тур неблагополучных районов

Описание: в каждом городе есть неблагополучные районы и места, которых туриstu лучше избегать. В программных продуктах тоже есть свои опасные места — секции кода, в которых полно багов.

Применение: главная цель разработчиков Chrome — сделать просмотр веб-страниц быстрым и простым. При этом насыщенный контент может страдать. В первых версиях Chrome у пользователей возникали проблемы даже с просмотром роликов на YouTube. Хотя многие проблемы в этом направлении уже решены, с функционально насыщенным контентом до сих пор бывают трудности.

Неблагополучные районы в Chrome OS

В тур неблагополучных районов в Chrome входят:

- **Онлайн-видео:** Hulu, YouTube, ABC, NBC, полноэкранный режим, высокое разрешение.
- **Flash-контент:** игры, реклама и презентации.
- **Расширения:** расширения, требующие усложненного тестирования.
- **Апплеты Java:** проверка успешного запуска апплетов Java (например, игр Yahoo!)
- **Технология O3D:** проверка контента, написанного с использованием технологии Google O3D (например, в видеосвязи Gmail).
- **Множественный запуск:** попытка открыть несколько экземпляров с «тяжелым» контентом в разных вкладках и окнах.

Тур персонализации

Описание: тур показывает путешественникам, как они могут персонифицировать свою поездку, подогнать ее под свои предпочтения. В тур все включено — от покупки новой пары солнцезащитных очков до аренды машины, найма личного экскурсовода и посещения магазинов. В тестировании наших приложений этот тур позволяет пользователям попробовать по-разному настроить свою работу с продуктом и персонализировать продукт на свой вкус.

Применение: исследование различных способов настройки Chrome для предпочтений конкретного пользователя с помощью тем оформления, расширений, закладок, настроек, ярлыков и профилей.

Способы настройки Chrome

В тур персонализации Chrome входят:

- **Темы:** использование тем для настройки внешнего вида Chrome OS.
- **Расширения:** загрузка и установка расширений Chrome OS для модификации функциональности и оформления.
- **Настройки Chrome:** настройка взаимодействия пользователя с Chrome с помощью изменения конфигурации.
- **Разделение профилей:** проверка того, что настройки одного пользовательского профиля не будут распространяться на другие учетные записи.

Посты из блога об инструментах и коде

В этом приложении приведено несколько постов из Google Testing Blog.

Охотимся на баги и потерянное время вместе с BITE

Среда, 12 октября 2011 г. 9:21

[http://googletesting.blogspot.com/2011/10/
take-bite-out-of-bugs-and-redundant.html](http://googletesting.blogspot.com/2011/10/take-bite-out-of-bugs-and-redundant.html)

Джо Аллан Мухарски

Хотя веб становится все более удобным и легким в работе, регистрация багов на сайтах — все еще ручной и утомительный труд. Найди дефект. Переключись на окно багтрекинговой системы. Заполни поля описания ошибки. Зайди обратно в браузер, сделай скриншот экрана и присоедини его к сообщению о баге. Не забудь добавить комментарий. Во время всего процесса приходится постоянно прыгать с одного контекста на другой. Внимание тестировщика постоянно отвлекается от приложения, которое он тестирует.

BITE (Browser Integrated Testing Environment) — расширение Chrome с открытым кодом (<http://code.google.com/chrome/extensions/index.html>), которое помогает упростить ручное веб-тестирование (рис. В.1). Чтобы расширение заработало, нужно связать его с сервером, на который он будет отправлять информацию о багах и тестах вашей системы. Тогда через BITE можно будет регистрировать баги в контексте веб-сайта, используя подходящие шаблоны.

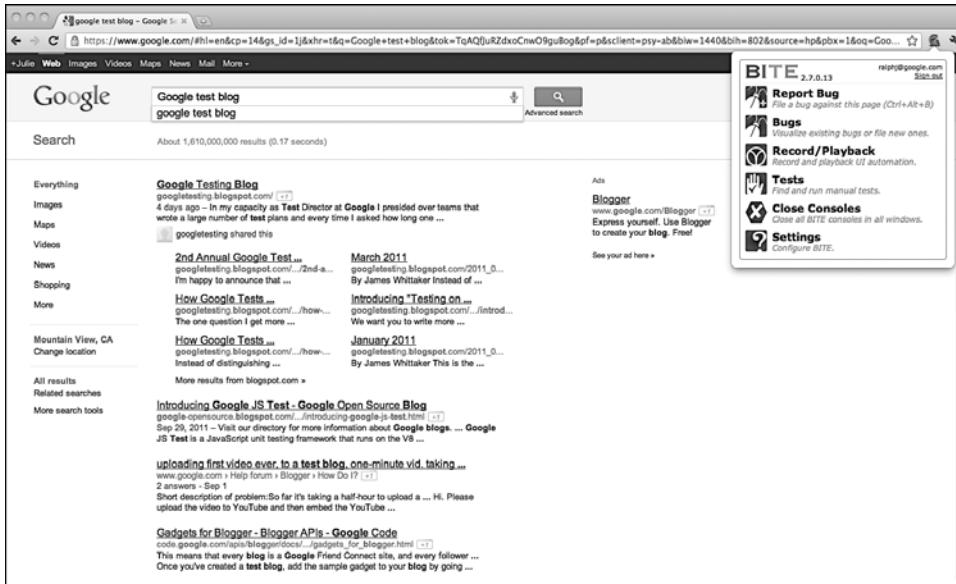


Рис. В.1. Меню расширения BITE в браузере Chrome

Когда вы регистрируете баг, BITE автоматически делает скриншот экрана, копирует ссылки и проблемные элементы интерфейса, а потом присоединяет их к описанию бага, как показано на рис. В.2. Теперь у разработчика, который анализирует или исправляет баг, будет достаточно информации, чтобы воспроизвести условия и найти причину нестандартного поведения.

Обычно тестировщикам приходится запоминать и дотошно фиксировать свои действия, чтобы баг можно было воспроизвести. Используя BITE, инженер может быть уверен, что все его ходы записаны в коде JavaScript и их можно посмотреть позже. Теперь можно быстро определить, воспроизводится ли баг в конкретной среде или решилась ли проблема при изменении кода.

Чтобы записывать действия при ручном тестировании и потом воспроизводить их при автоматизированном, в BITE есть консоль Record Playback Framework (RPF). Она автоматически генерирует код JavaScript, который потом можно использовать для воспроизведения выполненных шагов. Механизм записи и воспроизведения BITE устойчив к ошибкам: автоматизированные тесты пользовательского интер-

файса иногда могут не проходить, причем проблема часто в самом тесте, а не в продукте. При сбое воспроизведения BITE тестировщик может просто перезаписать свои действия на странице. Не нужно редактировать код или сообщать о сбое теста — если сценарий не может найти кнопку, которую нужно нажать, просто нажмите на нее сами, и сценарий исправится. Если все же без правки кода совсем не обойтись, мы используем редактор Ace (<http://ace.ajax.org/>), чтобы изменять код JavaScript в реальном времени.

Страница проекта BITE находится по адресу <http://code.google.com/p/bite-project>. Все вопросы и предложения присылайте по адресу bite-feedback@google.com.

Отправлено Джо Алланом Мухарски из команды технологий веб-тестирования (продукт был создан участниками команды Джейсоном Стредвиком, Джулией Ральф, По Ху и Ричардом Бустаманте).

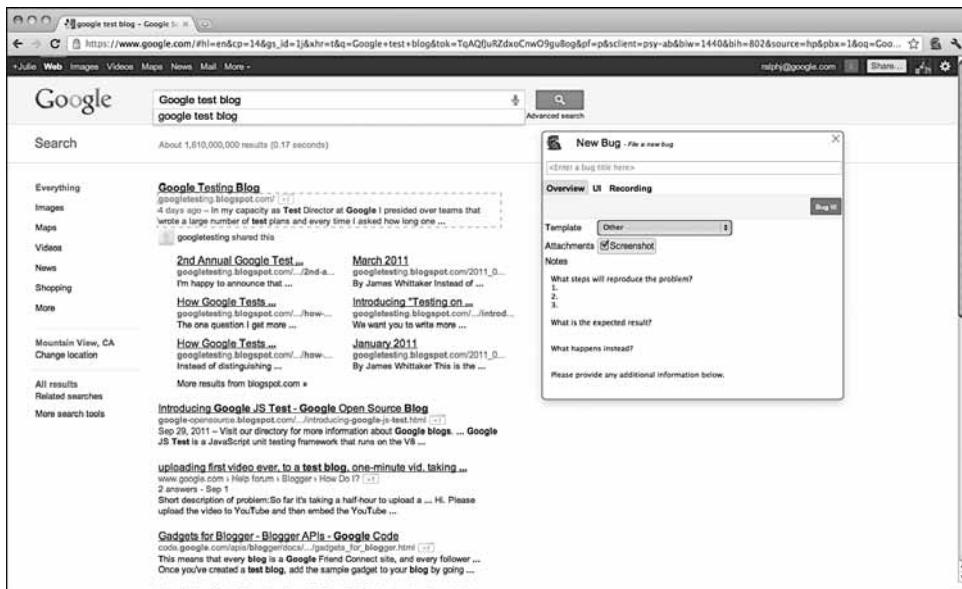


Рис. В.2. Интерфейс регистрации багов расширения BITE

QualityBots идет в атаку

Вторник, 06 октября 2011 г. 13:52

[http://googletesting.blogspot.com/2011/10/
unleash-qualitybots.html](http://googletesting.blogspot.com/2011/10/unleash-qualitybots.html)

Ричард Бустаманте

Вы разрабатываете веб-сайт и хотите знать, не сломают ли его работу обновления Chrome из дорелизных каналов? Вы хотите простой инструмент для сравнения работоспособности вашего сайта во всех каналах выпуска Chrome? Такой инструмент есть!

QualityBots (<http://code.google.com/p/qualitybots/>) — это новый инструмент с открытым кодом, который создала команда веб-тестирования Google для веб-разработчиков. Он сравнивает веб-страницы в версиях разных каналов Chrome с помощью попиксельного DOM-анализа. Когда появляется новая версия Chrome, QualityBots помогает нам рано находить сбои. Кроме того, QualityBots помогает разработчикам быстро и легко представить, как их страницы будут себя вести в версиях разных каналов Chrome.

Фронтенд-часть QualityBots построена на базе Google AppEngine (<http://code.google.com/appengine/>), а бэкенд, который обходит веб-страницы, работает на базе EC2. Чтобы использовать QualityBots, у вас должна быть учетная запись в Amazon EC2. Тогда вы сможете запускать виртуальные машины, которые будут обходить веб-страницы с разными версиями Chrome. У инструмента есть веб-интерфейс, в котором пользователи входят в систему, вводят URL-адреса страниц для обхода, просматривают результаты последнего запуска на информационной панели и подробно анализируют информацию о проблемных элементах страницы (как показано на рис. В.3).

С помощью этих результатов разработчики и тестировщики определяют сайты, которым нужно повышенное внимание из-за большого объема изменений. Если страница отображается практически идентично во всех каналах Chrome, ее можно пропустить (рис. В.4). Это экономит время и избавляет от утомительного тестирования сайтов, на которых ничего не изменилось.

Надеемся, разработчики веб-сайтов заинтересуются продуктом и присоединятся к проекту на странице QualityBots (<http://code.google.com/p/qualitybots/>). Будем рады обратной связи по адресу qualitybots-discuss@googlegroups.com.

Отправлено Ибрагимом Эль Фаром, команда технологий веб-тестирования (продукт был создан участниками команды Эриелом Томасом, Джейсоном Стредвиком, Ричардом Бустаманте и Теджасом Шахом).

Layout of page <http://finance.google.com> on Latest Run

Google finance

Market summary

European Leaders to Lay Out Plans to Contain Crisis

Financial markets rebounded sharply and investors expect that European leaders will do more to plug the sovereign debt crisis in the Eurozone. Wall Street jumped with DJIA and S&P 500 gaining +2.53% and +2.23% respectively. In the commodity sector, oil prices neared early losses and ...

Recent Quotes You have no recent quotes

Top stories Market

World stocks bounce on hopes for euro zone action

Reuters - 5 hours ago

1 of 2. A woman walks past a stock quotation board outside a brokerage in Tokyo September 26, 2011. By Alex Richardson SINGAPORE (Reuters) - Most CEOs are not good capital allocators when it comes to their are not objective analyzing their company and thus not objective in

Sales of new homes fall in August

Los Angeles Times - 5 hours ago

The sales rate is down 2.3% from July and up just 6.1% from a year ago, shaping up to be the worst year on record for new home sales.

United Airlines Posts Profit

Wall Street Journal - 7 hours ago

By SUSAN CAREY AND JACK NICAS CHICAGO—In the year since Airlines and Continental Airlines merged, the economy has weaker

PRECIOUS Gold stages comeback with aid of weak dollar

Reuters - 2 hours ago

Gold and silver bars are pictured at the Austrian Gold and Silver Se "Oegusai" in Vienna August 26, 2011. By Rujun Shen SINGAPORE (Reuters) -

More market news

Chrome 15.0.874.24

Enable Element Overlay

Google finance

Market summary

Asian stocks rebound on hopes for euro zone plan

Asian shares rebounded after the euro clung to gains on Tuesday on hopes that euro zone officials will act to ... Asian stocks end on hopes of euro zone resolution

Forbes

Asian stocks jump on hopes of euro debt resolution

Boston Globe

Financial Times TheStreet.com

Related articles

Top stories Market

PRECIOUS Gold stages comeback with aid of weak dollar

Reuters - 5 hours ago

Gold and silver bars are pictured at the Austrian Gold and Silver Se "Oegusai" in Vienna August 26, 2011. By Rujun Shen SINGAPORE (Reuters) -

Apple Report by JPMorgan Asia Staff Not View of U.S. Team, Mos Bloomberg - 7 hours ago

By Adam Lashinsky and Edmund Lococo - Tue Sep 27 03:29:42 G

Apple Inc. employee, left, gestures towards an iPad 2 during the q

United Airlines Posts Profit

Wall Street Journal - 5 hours ago

By SUSAN CAREY AND JACK NICAS CHICAGO—In the year since Airlines and Continental Airlines merged, the economy has weaker

Sales of new homes fall in August

Los Angeles Times - 5 hours ago

The sales rate is down 2.3% from July and up just 6.1% from a year ago, shaping up to be the worst year on record for new home sales.

Few shouts for Operation Twin

MarketWatch - 26 minutes ago

By Ian Williams PHOT WASHINGTON, NY (Market

Chrome 14.0.835.186 - baseline

Layout Score 83% Current

1071 elements total

1039 matched

32 need investigation

Chrome Channels

BASE	Stable	16.0.835.186
89	dev	16.0.891.0
80	canary	16.0.892.0
83	beta	15.0.874.24

Рис. В.3. Пример тестового запуска QualityBot. Результаты показывают различия в отображениях веб-сайтов в разных версиях

Results for Latest Run

Chrome Channels

Rel	Beta	Dev	Cnr	URL
○	○	○		http://www.pixlr.com
○	○	○		http://www.chatster.com
○	○	○		http://www.nasa.gov
○	○	○		http://www.chase.com
✗	✗	✗		http://www.washington.edu/
✗	✗	✗		http://books.google.com
✗	✗	✗		http://news.yahoo.com shared
○	○	○		http://news.google.com shared
○	○	○		http://www.youtube.com shared
✗	✗	✗		http://www.yahoo.com shared
✗	✗	○		http://www.ebay.com shared
✗	✗	○		http://finance.google.com shared
✗	●	●		http://www.nytimes.com shared
✗	✗	✗		http://www.hulu.com shared
○	○	●		http://www.google.com shared
○	○	●		http://www.xkcd.com shared

Layout Score Legend

- exact
- 99
- 89-50
- 49-0
- No Data

http://...

Рис. В.4. Информационная панель QualityBot с набором сайтов, промтестированных в разных версиях Chrome

RPF: Record Playback Framework

Вторник, 17 ноября 2011 г., 5:26

[http://googletesting.blogspot.com/2011/11/
rpf-googles-record-playback-framework.html](http://googletesting.blogspot.com/2011/11/rpf-googles-record-playback-framework.html)

Джейсон Арбон

На конференции GTAC меня спросили, насколько хорошо Record Playback Framework работает в среде BITE. Мы были настроены скептически, но подумали, что кто-то должен попробовать это оценить. Сейчас я расскажу вам, как взялись оценивать качество RPF и что из этого вышло.

Идея была в том, чтобы пользователи могли использовать приложение в браузере, записывать свои действия и сохранять их в виде кода JavaScript для воспроизведения в ходе регрессионных тестов. Как и большинство инструментов, основанных на генерации кода, RPF работает, но не идеален. У По Ху была ранняя рабочая версия, которую он решил опробовать на реальном продукте. По, разработчик RPF, работал с командой веб-магазина Chrome, на котором проверялась работоспособность ранней версии. Почему именно веб-магазин Chrome? Большая часть пользовательского интерфейса этого сайта управляется данными, на нем используется аутентификация, поддерживается возможность отправки файлов, к тому же сайт часто меняется и постоянно ломает существующие сценарии Selenium. Это был крепкий орешек для тестирования.

Прежде чем обратиться к разработчику тестов веб-магазина Chrome Венси Лю, мы сделали то, что нам казалось разумным: провели нечеткий поиск соответствий и построчно обновили тестовые сценарии. Selenium — отличная штука, но после создания стартового регрессионного пакета многие команды тратят кучу времени на простое сопровождение своих тестов, так как продукты постоянно меняются. Мы подумали: а что, если инструмент не будет просто проваливать тест, если не найдет нужный элемент, как это делает Selenium? А что, если нам не нужно будет анализировать DOM вручную и быстро менять код теста, а потом разворачивать его снова и запускать, и так по кругу? Что, если тестовый сценарий продолжит выполняться, а обновление кода сведется к простому клику на нужном элементе? Мы же можем брать значения атрибутов записанных элементов веб-страницы и, выполняя тест, просто вычислять процент их совпадений с теми, которые нашел тест во время выполнения. Если совпадение не стопроцентное, но лежит в допустимых пределах (например, изменился только родительский узел или атрибут класса), то мы запишем предупреждение и продолжим выполнение тест-кейса. Если следующие шаги теста проходят нормально, то тесты в сериях продолжают

выполняться и только записывают предупреждения. Если запустить тесты в режиме отладки — они приостановятся, чтобы тестировщик мог быстро обновить условия совпадения через пользовательский интерфейс BITE. Мы решили, что это сократит ложноположительные срабатывания и ускорит обновление.

Мы ошиблись, но в хорошую сторону!

Мы оставили тестировщика наедине с RPF, а потом поговорили с ним через несколько дней. Он уже воссоздал большинство своих тестовых пакетов Selenium в RPF, причем тесты уже начали ломаться из-за изменений в продукте (тестировщикам в Google нелегко угадаться за темпами поставки изменений разработчиками). Он выглядел вполне довольным, и мы спросили, как справляется новый хитрый механизм нечеткого поиска соответствий. «А, это? Даже не знаю. Я не пользовался...» — ответил Венси. Мы начали подозревать, что наш интерфейс обновления тестов был непонятным, не работал или его было трудно найти. Но Венси сказал, что для упавших ему тестов было проще записать сценарий заново. Все равно нужно повторно тестировать продукт, так почему бы не начать запись во время ручной проверки рабочего элемента, затем удалить старый тест и сохранить вновь записанный сценарий?

За первую неделю использования RPF Венси узнал следующее.

- 77% функций веб-магазина можно было тестировать в RPF.
- Генерация регрессионных тест-кейсов в этой, ранней версии RPF шла примерно в восемь раз быстрее, чем в Selenium/WebDriver.
- Сценарии RPF обнаружили шесть функциональных регрессионных багов и много непостоянных багов серверной части.
- Общие подготовительные шаги (например, вход в систему) должны сохраняться в виде модулей для повторного использования (простейшая версия этой функциональности вскоре заработала).
- RPF работает на Chrome OS, где Selenium не может работать в принципе, потому что требует наличия бинарных файлов на стороне клиента. RPF работает, потому что это облачное решение, которое полностью выполняется в браузере и взаимодействует с бэкенд-системой через веб.
- В созданных через BITE описаниях багов есть простая ссылка, которая устанавливает BITE на машине разработчика и воспроизводит запись на его машине. Программировать шаги записи вручную больше не нужно. Это очень удобно.
- Венси высказал пожелание, чтобы технология RPF работала не только для Chrome, ведь люди все-таки использовали разные браузеры для посещения сайтов.

Итак, мы поняли, что вырисовывается кое-что интересное, и продолжили разработку. Правда, тестирование веб-магазина Chrome скоро вернулось на технологию Selenium, потому что оставшиеся 23% функциональности требовали локального кода Java для поддержки отправки файлов и безопасного входа. Оглядываясь назад, мы понимаем, что небольшое улучшение тестируемости на сервере решило бы проблему с AJAX-вызовами на клиенте.

Мы проверили, как RPF ведет себя на популярных сайтах. Вся информация доступна в открытой вики проекта BITE. Сейчас она немного устарела, многое исправлено по сути, но можно сформировать общее представление о том, что не работало. Считайте, что показано качество альфа-версии. Итак, RPF работал для большинства сценариев, но оставались еще серьезные нестандартные случаи.

Джо Аллан Мухарски проделал огромную работу над нашим неуклюжим пользовательским интерфейсом, чтобы превратить его из чисто гиковской вещи для разработчиков в нечто интуитивно понятное. Он попробовал скрывать пользовательский интерфейс до тех пор, пока в нем не возникнет необходимость, и сделать все простым и удобным для поиска. Мы не проводили формальных исследований удобства использования, но мы наблюдали за сообществом тестировщиков, которые использовали эти инструменты почти без инструкций, и внутренними пользователями, которые сообщали о багах Google Maps без особых проблем. В более сложных частях RPF обнаружились неприятные сюрпризы, но базовая функциональность записи и воспроизведения, похоже, для большинства людей оказалась вполне понятной.

Со временем проект RPF вырос из экспериментального проекта команды тестирования в формальную часть всей команды Chrome, где его теперь используют для регрессионного тестирования. Команда работает над тем, чтобы дать возможность тестировщикам, которые не умеют программировать, возможность генерировать регрессионные сценарии с помощью BITE/RPF.

Присоединяйтесь к нашей работе по сопровождению BITE/RPF (<http://code.google.com/p/bite-project/>). Не забудьте поблагодарить По Ху и Джоэла Хиноски, которые двигают проект вперед.

Google Test Analytics – теперь с открытым кодом

Среда, 10 октября 2011 г., 13:03

[http://googletesting.blogspot.com/2011/10/
google-test-analytics-now-in-open.html](http://googletesting.blogspot.com/2011/10/google-test-analytics-now-in-open.html)

Джим Рирдон

Тест-план мертв!

Ну, мы на это надеемся. Неделю назад на семинаре STAR West Джеймс Уиттакер выяснил мнение профессиональных тестирующих о тест-планах. Его первый вопрос был такой: «Кто из присутствующих пишет тест-планы?». Поднялось около 80 рук — подавляющее большинство. «Кто из вас получает от тест-планов практическую пользу или возвращается к ним через неделю?» Руки подняли ровно три человека.

Мы тратим много времени на составление огромных документов, содержащих объемные описания подробностей проекта. Хотя все отлично знают, что скоро эти документы будут заброшены.

Мы в Google решили создать методологию на замену тест-планам. Она должна быть полной, быстрой, единственной и приносить постоянную пользу проекту. Не так давно Джеймс уже писал в нашем блоге об этой методологии, которую мы назвали ACC. У нас получился инструмент, который разбивает продукт на составные части, и метод, который помогает составлять «10-минутные тест-планы» (вообще-то это занимает около 30 минут!).

Полнота

Методология ACC создает матрицу, полностью описывающую ваш проект. С ее помощью в нескольких внутренних проектах Google нашли области покрытия, упущеные в ходе обычного планирования тестирования.

Скорость

Методология ACC работает быстро: создание классификации ACC даже в сложных проектах занимало у нас меньше получаса. Это намного быстрее составления тест-планов.

Действенность

В процессе анализа ACC вы связываете риски с возможностями вашего приложения. По этим значениям строится «тепловая карта» проекта, на которой видны области наибольшего риска, то есть места, которым следует уделить особое внимание при тестировании.

Польза

Мы создали несколько экспериментальных фич, которые воплощают в жизнь тест-план ACC, показывая данные, по которым можно количественно оценить риск в проекте (баги, тестовое покрытие).

Сегодня я с радостью объявляю, что мы открываем исходный код Test Analytics (<http://code.google.com/p/test-analytics/>) — инструмент, созданный в Google для упрощения ACC-анализа.

Test Analytics состоит из двух основных частей: первая часть — это пошаговый инструмент для создания матрицы ACC (рис. В.5). Он работает быстрее и проще электронных таблиц, которые мы использовали до этого (убедитесь, взглянув на рис. В.6). Наш инструмент визуализирует матрицу и риски, связанные с возможностями ACC, чем тоже не могли похвастаться обычные электронные таблицы (рис. В.7).

The screenshot shows the 'test analytics' application interface. The top navigation bar includes links for 'Simple Web Store', 'jim@amusive.com | Send Feedback | Sign out'. On the left, a sidebar menu lists 'Project Spec' (selected), 'About Project', 'Attributes' (selected), 'Components', 'Capabilities', 'Risk Overview', 'Imported Data', 'Tests', 'Bugs', 'Checkins', 'Data Settings', 'Data Sources', and 'Data Filters'. The main content area is titled 'Simple Web Store' and 'Attributes'. It contains a descriptive text: 'Attributes are ways a marketer would describe your application, such as "Fast" or "Secure". Attributes in the ACC model allow you to map Capabilities to business requirements.' Below this, there are three sections: 'Secure', 'Simple', and 'Fast'. Each section has a text input field labeled 'Enter description of this attribute...', a 'owner:' field containing 'margo@example', a 'tester:' field containing 'quentin@example', and a checkbox labeled 'The tests performed for this attribute are sufficient to verify its operation.' The 'Secure' section also has a note: 'This should be speedy.'

Рис. В.5. Определение атрибутов проекта в Test Analytics

testanalytics

jim@amusive.com | [Send Feedback](#) | [Sign out](#)

Simple Web Store ★ Simple Web Store

Project Spec
About Project
Attributes
Components
Capabilities
Risk Overview
Imported Data
Tests
Bugs
Checkins
Data Settings
Data Sources
Data Filters

Capabilities

Capabilities are what your application actually does; they are just like features, except they are tied to a specific Attribute and Component pair. [Learn more](#)

Capabilities by Attribute and Component

	Secure	Simple	Fast
Search	1	1	1
Social	2	1	1
Sales Channel	1	1	2
Shopping Cart	1	1	1

Social is Secure

User purchases not revealed outside granted permission.

User social graph not disclosed without permission.

Рис. В.6. Отображение возможностей проекта в Test Analytics

testanalytics

jim@amusive.com | [Send Feedback](#) | [Sign out](#)

Simple Web Store ★ Simple Web Store

Project Spec
About Project
Attributes
Components
Capabilities
Risk Overview
Imported Data
Tests
Bugs
Checkins
Data Settings
Data Sources
Data Filters

Known Risk

This shows the Total Risk to your application, taking into account any Risk Sources as well as Mitigation Sources that are checked below. [Learn more](#)

Inherent risk Bugs Code churn Test coverage

Risk displayed by Attribute and Component

	Secure	Simple	Fast
Search			
Social			
Sales Channel			
Shopping Cart			

Social is Secure

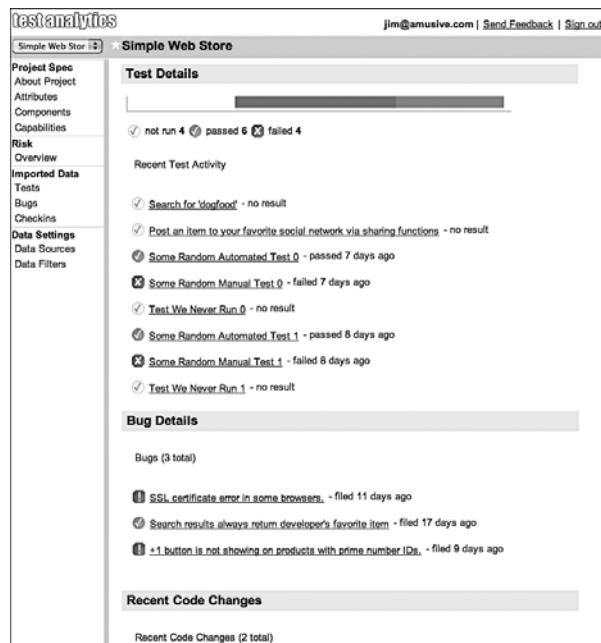
User purchases not revealed outside granted permission.

User social graph not disclosed without permission.

Рис. В.7. Отображение рисков в матрице атрибутов и компонентов в Test Analytics

Вторая часть Test Analytics превращает ACC-таблицу в живую, автоматически обновляемую матрицу рисков. Для этого Test Analytics импортирует из вашего проекта данные о качестве: баги, тест-кейсы, результаты тестов и изменения в коде. С этими данными риски становятся визуально понятными, основанными на количественных значениях, а не на догадках. Если некоторый компонент или возможность вашего проекта содержат многочисленные изменения в коде или открытые баги, то и риск в этой части выше. Проведенные тесты могут снизить риски. Если выполнить тесты и импортировать их результаты, показатели рисков в этой области снижаются по ходу тестирования.

Эта часть все еще остается экспериментальной. Мы пробуем разные способы вычисления риска по полученным результатам (см. рис. B.8). Тем не менее мы решили выпустить этот инструмент как можно раньше, чтобы узнать у сообщества, насколько хорошо наше решение подходит для команд тестировщиков. Мы хотим постоянно улучшать инструмент и делать его более полезным. Например, мы хотим импортировать больше информации о качестве: показатели сложности кода, результаты статического анализа кода, покрытие, обратная связь от внешних пользователей — это лишь часть идей, которые можно было бы включить в тест-план.



Rис. B.8. Test Analytics, привязанный к данным багов и тестовых примеров

Вы можете поэкспериментировать с живой версией продукта (<http://goo.gl/Cv2QB>), посмотреть код (<http://code.google.com/p/test-analytics/>) вместе с документа-

цией (<http://code.google.com/p/test-analytics/wiki/AccExplained>), и, конечно, мы будем рады вашим отзывам.

Для обсуждения продукта создана команда Google Group (<http://groups.google.com/group/test-analytics-discuss>), в которой мы активно отвечаем на вопросы и обмениваемся впечатлениями по поводу использования Test Analytics.

Тест-план мертв, да здравствует тест-план!

Дж. Уиттакер, Дж. Арбон, Дж. Каролло
Как тестируют в Google

Перевели с английского Алена Васюхина, Юлия Нечаева

Заведующий редакцией

Александр Кривцов

Руководители проекта

Юлия Нечаева,

Андрей Юрченко

Ведущий редактор

Юлия Сергиенко

Редактор

Юлия Нечаева

Научный редактор

Ксения Развенская

Дизайнеры

Артем Шитов,

Таня Черкиз

Иллюстрация на обложке

Максим Дегтярев

Корректоры

Надежда Викторова,

Вероника Ганчуриня

Верстка

Любовь Родионова