Testuj oprogramowanie jak

Metody automatyzacji

Poznaj najlepszą na świecie metodę testowania!

James Whittaker • Jason Arbon • Jeff Carollo

Helion

Tytuł oryginału: How Google Tests Software

Tłumaczenie: Julia Szajkowska

ISBN: 978-83-246-8656-8

Authorized translation from the English language edition, entitled:
HOW GOOGLE TESTS SOFTWARE; ISBN 0321803027; by James A. Whittaker;
and Jason Arbon; and Jeff Carollo; published by Pearson Education, Inc,
publishing as Addison Wesley.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/teopgo
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 1

Wprowadzenie do procedur testowania oprogramowania stosowanych przez firmę Google

James Whittaker

Jedno pytanie słyszę częściej niż jakiekolwiek inne. Niezależnie od tego, w jakim przebywam kraju czy w jakiej biorę udział konferencji, ono zawsze pojawia się prędzej czy później. Nawet Nooglerzy, jak nazywamy nowych pracowników korporacji Google, zadają je, gdy tylko nabiorą nieco praktyki w pracy. Brzmi ono: **„Jak Google testuje oprogramowanie?”**.

Nie potrafię powiedzieć, ile razy już na nie odpowiadałem ani nawet określić, ile różnych wersji odpowiedzi udzieliłem, wiem natomiast, że sama odpowiedź zmienia się — im dłużej pracuję dla Google, tym więcej dowiaduję się o niuansach technik kontrolowania jakości programów. Od dawna już ścigała mnie wizja napisania takiej książki, więc gdy Alberto, który zazwyczaj odgraża się, że przerobi każdą książkę dotyczącą testowania oprogramowania na pieluchy dla dorosłych (twierdząc, że wtedy istnienie takich podręczników byłoby chociaż trochę uzasadnione), zaproponował, żebym usiadł do pracy, nie miałem już żadnych wątpliwości — ta książka musiała powstać.

A mimo to zwlekałem. Główny problem polegał na tym, że zupełnie nie nadawałem się na autora takiego podręcznika. Przede wszystkim w samym Google pracuje wiele bardziej niż ja obeznanych z tym tematem osób, zatem chciałem, by mogły pierwsze zmierzyć się z tym zadaniem. Druga sprawa, która nie pozwalała mi podjąć się pisania z czystym sumieniem, to fakt, że jako kierownik grupy testującej

przeglądarkę Chrome i system Chrome OS (które to stanowisko przejął obecnie jeden z moich byłych podwładnych) miałem wgląd jedynie w niewielki fragment rozwiązań testowych stosowanych w firmie Google. Czekało mnie jeszcze wiele nauki.

Testowanie oprogramowania w Google leży w gestii większego, kierowanego centralnie działu, tak zwanej grupy wydajności projektowej (ang. *Engineering Productivity Group*). Grupa ta obejmuje programistów, zespoły testujące oraz zespoły odpowiedzialne za pracą nad kolejnymi wersjami oprogramowania, a do obowiązków jej członków należy testowanie programu na wszystkich poziomach — od poprawności funkcjonowania poszczególnych modułów do sprawdzenia testów rozpoznawczych. Zespoły testujące dysponują szerokim wachlarzem narzędzi i infrastrukturą pozwalającą testować działanie wielu znanych z sieci rozwiązań: wyszukiwarki, wyświetlania reklam, dodatkowych aplikacji, serwisu YouTube i innych, które kojarzy się z marką Google. Google zdołało pokonać wiele trudności związanych z tempem pracy i skalą działania, dzięki czemu mimo rozległej infrastruktury ciągle jesteśmy w stanie wprowadzać nowe aplikacje z werwą właściwą nowym firmom. Jak zauważył w przedmowie Patrick Copeland, swój sukces Google zawdzięcza przede wszystkim zespołom testującym.

Testowanie oprogramowania w Google leży w gestii większego, kierowanego centralnie działu, tak zwanej grupy wydajności projektowej.

Gdy w grudniu 2010 system operacyjny Chrome OS trafił wreszcie do odbiorców, a kierowanie zespołem przekazałem szczęśliwie jednemu ze swoich podwładnych, mogłem więc wreszcie poświęcić nieco więcej uwagi pozostałym produktom Google. Tak rozpoczęła się praca nad tą książką. Zanim jednak przystąpiłem do pisania, postanowiłem spróbować swoich sił bardziej kameralnie, na blogu[1], we wpisie „How Google Tests Software[2]" — reszta to już historia. Sześć miesięcy później książka była gotowa. Teraz wiem, że nie powinienem był zwlekać tak długo z przystąpieniem do pracy. W ciągu tamtych sześciu miesięcy dowiedziałem się więcej na temat procesów testowania w Google niż przez dwa lata pracy, a książka uzupełniła materiały szkoleniowe dla Nooglerów.

To nie jedyna pozycja poświęcona testowaniu oprogramowania przez wielkie firmy. Gdy pracowałem w Microsofcie, Alan Page, BJ Rollison i Ken Johnston napisali *How We Test Software at Microsoft*[3], której większa część bazowała na doświadczeniach własnych autorów. Microsoft wyznaczał wtedy standardy jakości w dziedzinie testowania oprogramowania. Polityka tej firmy sprawiła, że etap testów zyskał najwyższy priorytet w oczach najlepszych inżynierów świata. Testerzy pracujący

[1] *http://googletesting.blogspot.com/2011/01/how-google-tests-software.html*

[2] „Testuj oprogramowanie jak Google" — *przyp. tłum.*

[3] *Jak testujemy oprogramowanie w Microsofcie — przyp. tłum.*

wcześniej dla Microsoftu byli wręcz rozchwytywani przez innych pracodawców, a Roger Shrerman, pierwszy przełożony grupy testującej w Microsofcie, ściągał do Redmont wszystkich, którzy przejawiali zdolności w tej dziedzinie. To były złote czasy testów oprogramowania.

Wtedy firma wydała obszerną dokumentację całego procesu.

Nie trafiłem do Microsoftu na tyle wcześnie, by móc uczestniczyć w przygotowaniu tej publikacji, ale dostałem drugą szansę. Gdy trafiłem do Google, testy nad produktami zaczynały dopiero nabierać rozpędu. Grupa wydajności projektowej liczyła wtedy zaledwie kilkaset osób — dla porównania: w obecnych czasach liczba ta wzrosła do 1200. Problemy, o których Pat wspominał w przedmowie, właśnie odchodziły w niepamięć, a firma wchodziła w etap najbardziej dynamicznego rozwoju w swoich dziejach. Blog poświęcony zagadnieniom testowania oprogramowania w Google odwiedzały setki tysięcy osób miesięcznie, a GTAC[4] na stałe weszła do kalendarza imprez branżowych. Niedługo po moim przejściu do Google Patrick dostał awans i stał się bezpośrednim przełożonym przynajmniej kilkunastu innych szefów projektów i kierowników działów. Jeśli szukać źródeł nowej fali złotego wieku testowania oprogramowania, to z pewnością można zaliczyć do nich Google.

Oznacza to, że procedury testowania stosowane w korporacji Google również zasługują na obszerną relację. Cały kłopot polega na tym, że nie potrafię takowych zdawać. Jednocześnie Google słynie ze stosowania prostego i bezpośredniego podejścia do prac nad oprogramowaniem, może więc i ja powinienem przyjąć tę politykę.

W książce *Testuj oprogramowanie jak Google* znajdziesz odpowiedzi na pytania dotyczące tego, co to znaczy być testerem w Google czy w jaki sposób podchodzimy do zagadnień skali, złożoności czy powszechnego użycia przygotowywanych u nas programów. Znajdziesz tu informacje, których próżno szukać gdzie indziej, ale jeśli nie zaspokoją one Twojej ciekawości, pamiętaj, że w sieci znajdziesz mnóstwo innych. Wystarczy je „wygoogłać"!

Jednak to nie koniec tej historii, a wydaje mi się, że warto przedstawić ją całą. W każdym razie ja wreszcie dojrzałem do jej przedstawienia. Metody testowania oprogramowania w Google mogą z czasem stać się pewnego rodzaju standardem dla innych firm, gdyż coraz większa liczba producentów rezygnuje z zamykania swoich produktów na dyskach komputerów, odchodząc od tego modelu na rzecz wolności, jaką oferuje sieć. Nie spodziewaj się zatem znaleźć w tej książce wielu punktów stycznych z wydawnictwem sygnowanym przez Microsoft. Podobieństwa ograniczają się do liczby autorów — w obu przypadkach za powstanie książki odpowiadały trzy osoby — oraz ogólnej zbieżności tematów — w każdej z tych książek znajdziesz opis praktyk związanych z testowaniem oprogramowania, stosowanych w wielkiej firmie. Poza tymi dwoma aspektami książki te różnią się we wszystkim.

---

[4] GTAC, czyli Google Test Automation Conference (*https://developers.google.com/google-test-automation-conference/*), to organizowana co roku konferencja poświęcona zagadnieniom testowania oprogramowania.

Metody testowania oprogramowania w Google mogą z czasem stać się pewnego rodzaju standardem dla innych firm, gdyż coraz większa liczba producentów rezygnuje z zamykania swoich produktów na dyskach komputerów, odchodząc od tego modelu na rzecz wolności, jaką oferuje sieć.

Patrick Copeland w przedmowie przygotowanej do tej książki opisał, w jaki sposób narodziła się metodologia stosowana dziś w Google. Oczywiście jej początki, odpowiadające początkom samej firmy, stały się zrębem dla metod używanych dziś, które w naturalny sposób wykształciły się na tej podstawie wraz z rozwijaniem się samego Google. Należy mieć świadomość, że Google to tygiel umysłów ścisłych, miejsce spotkania ludzi, którzy szlify w fachu inżynierskim zdobywali w wielu innych firmach. Kultywowana w nim polityka innowacyjności sprawiła, że techniki, które nie sprawdziły się w innych przedsiębiorstwach, były przez pracowników Google odrzucane lub modyfikowane. Z czasem, gdy szeregi testerów w Google zaczęły się rozrastać, zaczęto wprowadzać nowe pomysły i wdrażać nieznane dotąd procedury. Te z nich, które sprawdziły się w praktyce w Google, wrosły na dobre w model testowy stosowany w firmie, a pozostałe — jako zbędny balast — odrzucono. Testerzy pracujący dla Google z chęcią dadzą szansę każdemu pomysłowi, ale bardzo szybko odrzucają te z rozwiązań, które nie sprawdzają się w specyficznych warunkach firmy.

Google to przedsiębiorstwo, w którym ceni się innowacyjność i szybkość działania i które słynie z tego, że udostępnia swoje oprogramowanie, gdy tylko kod nadaje się do użycia (i jednocześnie kiedy na ewentualne skutki błędów naraża się zaledwie garstkę użytkowników), a także ze zbierania opinii na temat funkcji oprogramowania już wraz z pierwszymi zainteresowanymi (co zwiększa ich liczbę i pozwala lepiej ocenić możliwości programu). Prowadzenie testów w takich warunkach musi przebiegać przede wszystkim sprawnie, zatem techniki wymagające wcześniejszego planowania czy stałego nadzoru zwyczajnie się nie sprawdzą. Zdarza się, że testowanie odbywa się jednocześnie z pracami nad oprogramowaniem, tak że często trudno jest określić, w którym miejscu przebiega granica między tymi dwoma dziedzinami. Z kolei w innych przypadkach testy są całkowicie niezależne od etapu programowania, więc twórcy aplikacji nie mają nawet świadomości, że ich produkt podlega badaniom.

Zdarza się, że testowanie odbywa się jednocześnie z pracami nad oprogramowaniem, tak że często trudno jest określić, w którym miejscu przebiega granica między tymi dwoma dziedzinami. Z kolei w innych przypadkach testy są całkowicie niezależne od etapu programowania, więc twórcy aplikacji nie mają nawet świadomości, że ich produkt podlega badaniom.

Rozwój Google spowodował oczywiście spadek tempa prac nad oprogramowaniem, ale w stopniu ledwie zauważalnym. Jesteśmy w stanie przygotować system operacyjny w czasie niewiele dłuższym niż rok, nowe wersje aplikacji klienckich, takich jak przeglądarki Chrome, pojawiają się raz na kilka tygodni, zaś aplikacje internetowe aktualizujemy codziennie — a wszystko mimo to, że referencje, którymi szczyciliśmy się na początku działania, dawno straciły już na aktualności. Wydaje się, że w przypadku tego środowiska znacznie łatwiej przyjdzie wskazać cechy, jakich brak prowadzonym w nim testom — dogmatyczności, sztywnych ram, wielkich nakładów pracy czy czasu — niż podać te, jakimi takie testy powinny się charakteryzować. Mimo to właśnie je postaram się określić. Jedno mogę stwierdzić z całą pewnością: testy nie mogą stanowić czynnika hamującego wprowadzanie innowacyjnych rozwiązań czy spowalniającego tempo prowadzenia prac nad aplikacją. W każdym razie żadna procedura testowania nie zdoła zrobić tego dwukrotnie.

Doskonałe wyniki otrzymywane dzięki metodom testowania stosowanym w Google z pewnością nie są skutkiem małej liczby oferowanych przez firmę aplikacji. Zakres i stopień złożoności testów prowadzonych w Google są takie same jak w każdej innej dużej firmie. Niezależnie od tego, czy będziemy mówić o systemach operacyjnych przeznaczonych dla urządzeń indywidualnych, aplikacjach internetowych, programach działających na urządzeniach mobilnych, czy aplikacjach wielostanowiskowych, rozwiązaniach dla biznesu lub sieci społecznościowych, z pewnością znajdziemy wśród nich towary z oferty Google. Nasze aplikacje są ogromne i niebywale złożone, korzystają z nich miliony użytkowników, bywają też celem ataków hakerskich. Spore partie kodu są powszechnie dostępne, wiele fragmentów stanowi schedę po stosowanych dawniej rozwiązaniach, a sama firma podlega częstym kontrolom. Oferowane przez nas programy działają w setkach krajów na całym świecie, funkcjonując oczywiście w różnych wersjach językowych, a gdyby tego było mało, zawsze można wspomnieć o podstawowym wymogu, jaki stawiają im użytkownicy — aplikacje Google powinny dać się łatwo używać i „po prostu działać". Z pewnością nie można powiedzieć, by praca testera w Google polegała na rozwiązywaniu banalnych problemów. Nasi testerzy przypuszczalnie muszą codziennie mierzyć się z każdym wyzwaniem, jakie można sobie wyobrazić.

To, czy Google wywiązuje się dobrze ze stawianych sobie zadań (pewnie nie), to kwestia dyskusyjna, natomiast z pewnością można powiedzieć jedno — stosowane w tej firmie podejście do zagadnienia testowania oprogramowania różni się wyraźnie od procedur, które miałem okazję obserwować w innych przedsiębiorstwach, a sądząc po postępującej nieubłaganie tendencji do przenoszenia oprogramowania z komputerów stacjonarnych do chmury, wydaje się, że praktyki stosowane w Google wkrótce staną się powszechne w tej branży. Mamy nadzieję, że wiedza zawarta w tej książce pozwoli rzucić nieco światła na rozwiązania stosowane w Google i tym samym doprowadzić do konstruktywnej dyskusji w temacie działań, jakie należałoby podjąć, by twórcy oprogramowania mogli oferować odbiorcom solidne, niezawodne produkty, na których ci mogliby polegać. Oczywiście pomysły wdrażane w Google nie są pozbawione wad, ale chcielibyśmy przedstawić je światu i poddać ocenie

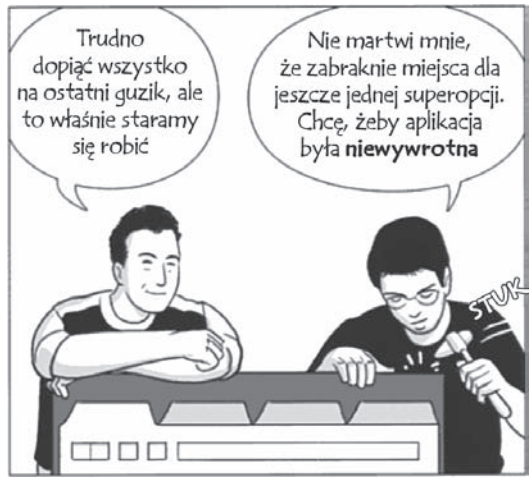
międzynarodowego środowiska zajmującego się zagadnieniem prowadzenia testów oprogramowania. Dzięki temu będziemy mogli dalej rozwijać je i poprawiać.

Reguły testowania w Google wydają się kłócić z nakazami logiki — w całej firmie pracuje mniej testerów niż w niejednym przedsiębiorstwie nad poszczególnymi projektami. Google Test to nie liczące miliony oddziały piechoty. Jesteśmy niewielkim i doborowym oddziałem specjalnym, którego podstawę działania stanowią starannie przygotowana taktyka i zaawansowane technologicznie uzbrojenie — tylko one dają nam szansę powodzenia w podejmowanych misjach. Brak potężnego zaplecza ludzi zmusza nas do konkretnego określania priorytetów. Larry Page ujął to doskonale: „Niedostatek podnosi przejrzystość". Nieważne, czy mówimy tu o funkcjach oprogramowania, czy metodach prowadzenia testów — doświadczenie nauczyło nas, że w obydwu przypadkach najlepsze wyniki uzyskuje się, stosując wysoce wydajne, ale nie nazbyt oporne metody działania. Również dzięki niedostatkom nauczyliśmy się cenić zasoby wykorzystywane do prowadzenia testów. Dlatego też doceniamy każdego pracownika i staramy się, by zatrudniani w Google błyskotliwi testerzy angażowali się całymi sobą w prowadzone prace. Gdy ktoś pyta mnie o klucz do sukcesu, odpowiadam zazwyczaj: „Nie zatrudniaj zbyt wielu osób w dziale testów".

Gdy ktoś pyta mnie o klucz do sukcesu, odpowiadam zazwyczaj: „Nie zatrudniaj zbyt wielu osób w dziale testów".

Jak zatem firma taka jak Google radzi sobie z tak małym zespołem testującym? Gdybym miał ująć to najprościej, jak potrafię, powiedziałbym, że w Google ciężar dbania o odpowiednią jakość oprogramowania spoczywa przede wszystkim na barkach programistów. Jakość nie jest nigdy problemem „tych tam, testerów". Każdy pracujący w Google programista jest jednocześnie testerem, więc o jakość dba dosłownie kolektyw (rysunek 1.1). Mówienie o stosunku zatrudnienia programistów do testerów w Goolge ma mniej więcej taki sam sens jak rozważanie stopnia zanieczyszczenia atmosfery przy powierzchni Słońca. Takie rozważania nie mają zwyczajnie sensu. Jeśli jesteś inżynierem, musisz zajmować się też testowaniem. Jeśli zaś jesteś inżynierem ze słowem „testowanie" w tytule, zdołasz nauczyć czegoś kolegów, którzy są tylko programistami.

To, że tworzymy oprogramowanie klasy światowej, dowodzi stanowczo, że wdrożone w Google rozwiązania zasługują na bliższe poznanie. Być może niektóre z naszych pomysłów sprawdzą się także w innych firmach, jednak z całą pewnością można stwierdzić, że część z nich wymaga ulepszenia. W dalszej części rozdziału znajdziesz skrócony opis metod testowania stosowanych w Google, a w pozostałych postaram się opisać, w jaki sposób usiłujemy łączyć praktyki testowe z tworzeniem kodu.



RYSUNEK 1.1. W Google stawiamy na jakość, a nie na wymyślne funkcje

Jakość ≠ testy

„Jakości nie da się wdrożyć za pomocą testów". To truizm, ale warto o tym pamiętać. Nieważne, czy mówimy o samochodach, czy o aplikacjach komputerowych — jeśli o jakość nie zadbamy już w fazie projektu, produkt nigdy nie *będzie* dobry. Wystarczy posłuchać skarg na koszty tych producentów samochodów, którzy musieli wprowadzać poprawki do ostatecznej wersji produktu. Jeśli to, nad czym pracujesz, nie będzie przygotowane starannie od samego początku, przygotuj się na poważne kłopoty.

Niestety nie jest to ani tak proste, jak mogłoby się wydawać, ani zawsze skuteczne. Choć jakości rzeczywiście nie wdraża się za pomocą testów, praktyka dowodzi, że bez testów nie da się osiągnąć odpowiedniej jakości. Bo jak bez sprawdzania ocenić, czy produkt jest wystarczająco dobry, by przekazać go w ręce klientów?

Podejdźmy do problemu jak do zagadki logicznej. Najprostsze rozwiązanie zakłada rezygnację ze stanowiska, że prace nad produktem i testowanie są od siebie niezależne. Testowanie i opracowywanie powinny przebiegać jednocześnie. Napisz fragment kodu i sprawdź, jak działa. Dopisz kolejny kawałek i znów przetestuj. Testowanie nie gwarantuje jakości. Jakość osiąga się, łącząc pracę nad kodem z ciągłą weryfikacją wyników — należy mieszać je ze sobą tak długo, aż staną się jednolitą masą.

Testowanie nie gwarantuje jakości. Jakość osiąga się, łącząc pracę nad kodem z ciągłą weryfikacją wyników — należy mieszać je ze sobą tak długo, aż staną się jednolitą masą.

To właśnie staramy się osiągnąć w Google — połączyć pracę nad kodem i testowanie go, tak by jedno nie istniało bez drugiego. Napisz coś, a potem to sprawdź. Napisz następną część i znowu sprawdź. W takim modelu najważniejsze jest to, *kto* przeprowadza testy. Ponieważ liczba faktycznych testerów w Google jest znacznie mniejsza niż liczba programistów, to właśnie na tych ostatnich spada obowiązek sprawdzania kodu. A kto lepiej poradzi sobie z oceną jakości zaimplementowanych rozwiązań, jeśli nie osoba odpowiedzialna za powstanie kodu? Kto szybciej znajdzie błąd w programie niż jego twórca? Google może pozwolić sobie na ograniczenie liczby weryfikatorów, ponieważ zatrudnia najwyższej klasy programistów. Jeśli aplikacja zawodzi, winą obarcza się wtedy przede wszystkim programistę, który popełnił błąd, a nie testera, który go nie odnalazł.

Oznacza to, że jakość wynika głównie z zapobiegliwości, a nie z działań mających na celu wykrywanie błędów. Jakość wynika bezpośrednio ze sposobu rozwijania projektu, a nie z metod, za pomocą których jest on testowany. Staraliśmy się stworzyć proces wielostopniowy — na tyle, na ile mogliśmy scalić testowanie z pracą nad kodem — dzięki czemu wszelkie błędy pojawiające się na danym etapie prac nad projektem można bardzo łatwo wycofać. Dzięki temu nie dość, że oszczędzamy wielu przykrych niespodzianek naszym klientom, to jeszcze udało nam się znacznie zmniejszyć liczbę osób niezbędnych do usuwania błędów wywołania. Ogólnie staramy się sprowadzić kwestię testowania do badania, na ile sprawdzają się procedury zapobiegania występowaniu błędów.

Polityka łączenia prac nad kodem z testowaniem jest nierozerwalnie związana ze sposobem myślenia preferowanym w Google. Na pytanie: „A gdzie wyniki testów?” można natknąć się u nas wszędzie — począwszy od uwag na marginesach raportów, po ściany toalet, gdzie umieszczamy plakaty przypominające naszym programistom o konieczności testowania kodu[5]. Testowanie musi być nieodzownym aspektem tworzenia kodu, gdyż dopiero mariaż procedur weryfikujących i programowania pozwala uzyskać produkt odpowiedniej jakości.

Testowanie musi być nieodzownym aspektem tworzenia kodu, gdyż dopiero mariaż procedur weryfikujących i programowania pozwala uzyskać produkt odpowiedniej jakości.

Podział ról

Aby pozostać w zgodzie z mottem: „Sam zrobiłeś, sam rozbierz” (i utrzymać ten stan rzeczy w miarę upływu czasu), poza utrzymywaniem typowego zespołu do zadań programistycznych firma musi określić dla pracowników także zakres innych obowiązków. Chodzi tu przede wszystkim o określenie czynności, które pozwolą

[5] *http://googletesting.blogspot.com/2007/01/introducing-testing-on-toilet.html*

programistom skutecznie i wydajnie testować programy. W Google zadbaliśmy o to, by niektórzy z naszych inżynierów mieli za zadanie nadzorować pracę innych, dzięki czemu ci drudzy osiągają lepsze wyniki w krótszym czasie. Grupa nadzorująca bardzo często określa się sama mianem *testerów*, ale jej podstawowym zadaniem jest czuwanie nad utrzymaniem wydajności pracy. Rolą testerów jest podnoszenie wydajności w zespołach programistycznych, a unikanie wprowadzania kolejnych poprawek, które muszą pojawiać się w kiepsko opracowanym kodzie, to bardzo istotny czynnik wydajności. Dlatego też w dalszych rozdziałach poświęcimy sporo miejsca na dokładne omówienie stosowanego w Google podziału ról. Na razie zaś przedstawiam krótkie ich zestawienie.

Inżynier oprogramowania (IO) to zwyczajny programista. Do jego obowiązków należy przygotowanie kodu, który ostatecznie trafia do użytkownika. To programiści przygotowują dokumentację, określają struktury danych projektu i jego architekturę, a większość czasu poświęcają na tworzenie kodu i sprawdzanie jego poprawności. Inżynier oprogramowania przygotowuje ogromne ilości kodu testującego, głównie w ramach pracy w systemie projektowania z użyciem testów (ang. *test-driven design*, TDD), tworzy testy jednostkowe i — co jeszcze wyjaśnimy w tym rozdziale — bierze udział w opracowaniu małych, średnich i wielkich procedur testujących. Inżynier oprogramowania odpowiada za jakość wszystkiego, z czym ma styczność, niezależnie, czy to oprogramował, wyrugował potknięcia innych, czy całkowicie zmodyfikował kod. Tak właśnie — jeśli inżynier wprowadza zmiany w istniejących funkcjach i przez to funkcja przestaje przechodzić przez istniejące już testy albo wymaga dodatkowej weryfikacji, to inżynier ma obowiązek przygotować nowe procedury sprawdzające. Inżynier oprogramowania poświęca się niemal całkowicie pisaniu kodu.

Inżynier do spraw testowania oprogramowania (ITO) to także stanowisko programistyczne, przy czym osoby je piastujące mają za zadanie skupiać się przede wszystkim na kwestii testowania i ogólnych założeniach procedur testujących. Do ich obowiązków należy opiniowanie projektów i badanie kodu pod kątem spełniania norm jakościowych oraz oceny ryzyka. Do ich zadań należy refaktoryzacja kodu, tak by dawał się on łatwiej testować, oraz przygotowywanie szkieletu testującego poszczególne moduły i automatyzowanie procesu sprawdzania jakości kodu. Ludzie zatrudnieni na tego rodzaju stanowiskach to także programiści, ale ich zainteresowanie skupia się przede wszystkim na podnoszeniu jakości produktu i prowadzeniu weryfikacji. Kwestie dodawania nowych funkcji czy zwiększania wydajności są dla nich drugorzędne. Choć większość czasu również pochłania im programowanie, to przygotowywany przez nich kod ma przede wszystkim podnosić jakość oferowanego produktu, a nie rozwijać istotne z punktu widzenia użytkownika funkcje.

Choć większość czasu również pochłania im programowanie, to przygotowywany przez nich kod ma przede wszystkim podnosić jakość oferowanego produktu, a nie rozwijać istotne z punktu widzenia użytkownika funkcje.

Inżynier testujący (IT) to stanowisko zbliżone do stanowiska inżyniera do spraw testowania oprogramowania, przy czym osoby je piastujące powinny skupiać się na nieco innym aspekcie prowadzenia testów. Inżynier testujący powinien badać produkt przede wszystkim z punktu widzenia użytkownika, a dopiero potem zastanawiać się, jak na ten sam problem spojrzy programista. Niektórzy z zatrudnionych w Google inżynierów testujących poświęcają mnóstwo czasu na przygotowanie kodu skryptów automatyzujących czy aplikacji realizujących różne scenariusze użytkowania, a czasami wręcz udających działanie użytkownika. Jednocześnie organizują testy prowadzone przez IO czy ITO, interpretują wyniki i przeprowadzają ostateczną weryfikację, szczególnie w końcowych fazach pracy nad projektem, gdy wysiłki wszystkich skupiają się na przygotowaniu produktu do wypuszczenia na rynek. Inżynierowie testujący to eksperci do spraw produktu, doradcy do spraw jakości i analitycy ryzyka. Wielu z nich zajmuje się aktywnie programowaniem, lecz równie liczna grupa praktycznie tego nie robi.

Uwaga

Inżynier testujący sprawdza program przede wszystkim z punktu widzenia użytkownika. Do jego zadań należy przygotowanie wszystkich działań związanych z podniesieniem jakości produktu, interpretowanie wyników testów, przeprowadzanie samych testów i tworzenie kompleksowej automatyzacji testów.

Wracając jednak do rozważań na temat jakości — inżynierowie oprogramowania zajmują się pewnymi aspektami działania aplikacji i kwestią jakości wdrażanych rozwiązań w raczej izolowanym środowisku. Do ich obowiązków należy przygotowanie projektu odpornego na usterki, przywracanie aplikacji do stanu sprzed pojawienia się błędu, projektowanie z użyciem testów, tworzenie testów jednostkowych i aktywna współpraca z inżynierami do spraw testowania oprogramowania podczas pisania kodu testującego funkcje badanej aplikacji.

Z kolei inżynierowie do spraw testowania oprogramowania to programiści przygotowujący funkcje testowe. Oni opracowują szkielet pozwalający wyizolować przygotowywany kod przez symulowanie środowiska, w jakim ma on docelowo działać (tu wykorzystywane są takie narzędzia, jak *stub*, *mock* i *fake* — metody, które opiszę dokładniej nieco dalej), i określanie kolejności, w jakiej kod będzie weryfikowany. Innymi słowy, inżynier do spraw testowania oprogramowania przygotowuje kod, dzięki któremu inżynier oprogramowania będzie mógł sprawdzić poprawność przygotowanych funkcji. Samo prowadzenie testów bardzo często spada już na IO. ITO ma za zadanie sprawdzić, czy poszczególne funkcje w ogóle nadają się do testowania, i dopilnować, by IO aktywnie uczestniczył w przygotowywaniu kodu przypadków testowych.

Z powyższego opisu wynika wyraźnie, że ITO bierze pod uwagę przede wszystkim perspektywę programisty. Jego zadaniem jest zadbać o jak najwyższą jakość

działania poszczególnych funkcji programu i jak najbardziej ułatwić programistom zweryfikowanie przygotowanego przez nich kodu. Prowadzenie testów uwzględniających punkt widzenia użytkownika to zadanie zatrudnianych przez Google inżynierów testujących. Gdy testy na poziomie modułowym i na poziomie funkcjonalności przeprowadzone przez IT oraz ITO dadzą zadowalające wyniki, nadejdzie pora, by sprawdzić, w jaki sposób użytkownik odczuje działanie aplikacji zasilanej odpowiednimi danymi. Testy wykonywane przez IT są poniekąd podwójnym sprawdzeniem sprawności programistów. Wszystkie wykryte przez nich oczywiste błędy będą dowodem, że weryfikacja wykonana na wczesnym poziomie przez programistów została przeprowadzona niestarannie i w nieodpowiedni sposób. Gdy okaże się, że takie błędy są sporadyczne, IT może zająć się podstawowym stawianym przed nim zadaniem, czyli sprawdzeniem, czy oprogramowanie będzie działać poprawnie w typowych warunkach użytkowania, czy będzie spełniać oczekiwania użytkownika, czy jest odpowiednio zabezpieczone, zlokalizowane, czy jest przystępne i tak dalej. Inżynier testujący zajmuje się przede wszystkim przeprowadzaniem testów i koordynowaniem pracy swoich kolegów oraz testerów kontraktowych, testerów „ z tłumu", testerów wewnętrznych (tzw. dogfooderów[6]), użytkowników wersji beta i awangardy wśród użytkowników. To właśnie inżynierowie testujący przekazują członkom zainteresowanych grup informacje o problemach mogących wystąpić w podstawowym projekcie, opisują im stopień skomplikowania funkcji aplikacji i metody unikania błędów. Trzeba przyznać, że praca inżyniera testującego nie ma końca.

Struktura organizacyjna

W większości firm, z którymi miałem okazję współpracować, programiści i testerzy tworzą jeden zespół pracujący nad danym projektem — i jedni, i drudzy odpowiadają przed tym samym szefem zespołu. Jeden produkt, jeden zespół i wszyscy stoją po jednej stronie barykady.

Niestety nigdy nie widziałem, żeby ten model sprawdził się w praktyce. Starsi menedżerowie są rekrutowani najczęściej spośród kierowników zespołów lub programistów — bardzo rzadko z grupy testerów. Stąd w gorącym okresie poprzedzającym premierę główny nacisk kładzie się raczej na dopracowanie funkcji aplikacji, a nie na testowanie jakości jej jądra. W tak zorganizowanym zespole testowanie spada na podrzędną pozycję, czego dowody znajdziemy zresztą w nie tak odległej przeszłości, pełnej wadliwych produktów i przedwczesnych wejść na rynek. Czy znajdę chętnych na Service Pack1?

[6] Określenie *dogfood* przyjęło się w większości firm informatycznych w Stanach Zjednoczonych. Opisuje ono proces adaptowania oprogramowania przygotowywanego do wprowadzenia na rynek wewnątrz danej firmy. Wyrażenie *eating your own dogfood* (zjadanie karmy swojego psa — *przyp. tłum.*) ma oddawać koncepcję, że zanim sprzeda się komuś własny produkt, należy sprawdzić jego jakość na sobie.

Uwaga

Mimo że zespół pracujący nad danym projektem powinien być zgrany, nadzorujący pracę wywodzą się zazwyczaj z grupy kierowników niższego szczebla lub projektantów, rzadko zaś spośród testerów. W gorącym okresie poprzedzającym premierę aplikacji główny nacisk kładzie się raczej na dopracowanie funkcji aplikacji, a nie na testowanie jakości jej jądra. To struktura organizacyjna sprawia, że testy zawsze będą traktowane z mniejszym namaszczeniem niż programowanie.

Strukturę organizacyjną Google tworzą tak zwane obszary zainteresowania (ang. *Focus Area*), czyli OZ-y. Osobny OZ obsługuje strefę klienta (Chrome, Google Toolbar i im podobne), osobny geolokalizację (Mapy, Google Earth itp.), a jeszcze inne zajmują się reklamami, aplikacjami czy programami na urządzenia mobilne. Wszyscy inżynierowie oprogramowania odpowiadają przed kierownikiem lub wiceprezesem danego OZ.

Inżynierowie do spraw testowania oprogramowania i inżynierowie testujący wyłamują się z tego schematu. Do testów dochodzi w niezależnej jednostce, której kompetencje przenikają przez wszystkie obszary zainteresowania, czyli we wspomnianej już grupie wydajności projektowej. Testerzy są wypożyczani poszczególnym zespołom programistów i nikt nie dziwi się, gdy zgłaszają uwagi dotyczące jakości produktu czy zastrzeżenia dotyczące tych jego funkcji, które nie zostały należycie przetestowane czy też które wykazują zbyt wiele błędów, by można było uznać je za dopuszczalne. Kierujemy się własnymi priorytetami i nie zarzucimy dbania o solidność rozwiązań czy bezpieczeństwo na rzecz niczego, co nie będzie naprawdę istotne. Jeśli zespołowi programującemu zależy na skróceniu testów, musimy wiedzieć o tym wcześniej, ale nawet gdy pojawi się taka prośba, zawsze mamy prawo odmówić.

Dzięki temu Google może ograniczyć liczbę testerów. Kierownik zespołu pracującego nad przygotowaniem aplikacji nie może samodzielnie zadecydować o obniżeniu wymagań jakościowych stawianych produktowi, nie może też zatrudnić większej liczby testerów, by zepchnąć na nich niewdzięczne zadania. Programista zajmujący się przygotowaniem danej funkcji programu musi zmierzyć się też z tymi mniej przyjemnymi i żmudnymi aspektami pracy nad aplikacją i naprawdę nie ma prawa spychać tych zadań na jakiegoś nieszczęsnego testera. O przydziale testerów do poszczególnych projektów decyduje kierownictwo grupy wydajności projektowej. Podstawą do podejmowania decyzji są priorytet przypisany danemu projektowi, złożoność zagadnienia i potrzeby danego zespołu oceniane w stosunku do potrzeb pozostałych grup. Oczywiście i my możemy się mylić — zresztą czasami tak właśnie się dzieje — ale mimo to zastosowane podejście pozwala wykorzystać siły, jakimi dysponujemy, do zaspokojenia prawdziwych, a nie wydumanych potrzeb.

Uwaga

Przydzielaniem testerów do poszczególnych projektów zajmuje się kierownictwo grupy wydajności projektowej. Podstawą do podejmowania decyzji są priorytet przypisany danemu projektowi, złożoność zagadnienia i potrzeby danego zespołu oceniane w stosunku do potrzeb pozostałych grup. Zastosowane podejście pozwala wykorzystać siły, jakimi dysponujemy, do zaspokojenia prawdziwych, a nie wydumanych potrzeb.

Takie „wypożyczanie" testerów sprawia też, że zadania stawiane przed inżynierami do spraw testowania oprogramowania i inżynierami testującymi ciągle się zmieniają, dzięki czemu ITO i IT nie tracą zainteresowania tematem, który przecież zawsze jest nowy i fascynujący. Jednocześnie rozwiązanie to zapewnia w naturalny sposób wzmożony przepływ pomysłów wewnątrz firmy. Tak zwiększamy prawdopodobieństwo ponownego wykorzystania dobrych procedur testowych — tester, który sprawdził jakąś koncepcję w czasie pracy nad projektem z działu geolokacji, użyje jej zapewne także po przeniesieniu go do prac nad przeglądarką Chrome. Nic tak nie przyspiesza rozprzestrzeniania się idei jak przeniesienie jej twórcy w nowe środowisko.

Przyjmujemy zasadniczo, że po osiemnastu miesiącach tester może (ale nie musi) zażądać przeniesienia do nowego projektu bez żadnych konsekwencji. Z jednej strony oznacza to oczywiście, że projekt traci eksperta, lecz jednocześnie pamiętajmy, że w konsekwencji stosowanej polityki testerzy Google znają się ogólnie na wszystkim i mają bogate doświadczenie w pracy nad różnymi rodzajami aplikacji. Można powiedzieć, że Google to firma zatrudniająca testerów, którzy rozumieją potrzeby klienta, znają możliwości sieci i przeglądarek, wiedzą, czego wymaga praca z technologiami mobilnymi, i potrafią pisać programy w wielu językach i na różne platformy. A ponieważ nasze aplikacje i usługi są teraz powiązane ze sobą ściślej niż kiedykolwiek w przeszłości, tester zmieniający zespół tak czy inaczej dysponuje odpowiednim doświadczeniem.

Od raczkowania, przez chód, do biegu

Google osiąga tak zdumiewające wyniki mimo zatrudniania mniejszej niż w innych firmach liczby testerów, ponieważ w odróżnieniu od innych dużych firm rzadko usiłujemy wprowadzać do naszych produktów mnóstwo innowacji naraz. W zasadzie działamy zupełnie inaczej — najpierw przygotowujemy starannie jądro aplikacji i w chwili, gdy staje się ono zdatne do użycia, staramy się udostępnić je jak najszerszej grupie odbiorców. Następnie zapoznajemy się z ich uwagami i próbujemy wprowadzić je w życie, a gdy poprawki są gotowe, znów udostępniamy je klientom. Tak postąpiliśmy w przypadku aplikacji Gmail, która przez cztery lata funkcjonowała formalnie jako wersja beta. Etykietka pojawiająca się przy nazwie usługi ostrzegała

użytkowników, że produkt jest ciągle w fazie ulepszeń. Usunęliśmy ją dopiero, gdy udało się nam osiągnąć zamierzony wcześniej cel — serwery obsługujące rzeczywiste konta użytkowników przez 99,99% czasu. Podobnie postąpiliśmy z Androidem, udostępniając go wyłącznie na telefonie G1, które to połączenie zyskało sobie uznanie użytkowników i recenzentów. Nexus, telefon nowej linii i następca G1, funkcjonował już z nową wersją systemu — o bardziej rozbudowanych funkcjach. Tu należy podkreślić, że gdy klient płaci za wczesną wersję towaru, musi ona być funkcjonalna na tyle, by nie miał on poczucia, że został oszukany. To, że jest to wczesna wersja, nie musi oznaczać, że towar nie nadaje się do użycia.

Uwaga

Google często wprowadza na rynek „najmniej rozbudowaną działającą wersję" programu i dopiero z czasem rozwija ją o kolejne możliwości, zbierając jednocześnie wrażenia użytkowników — tak tych wewnątrz firmy, jak i pierwszych klientów. Przed wykonaniem kolejnego niewielkiego kroku naprzód zawsze starannie rozważamy jego wpływ na jakość produktu. Każda aplikacja Google przechodzi przez kilka etapów: wersję „kanarkową" (ang. *canary*), deweloperską, testową, beta i RC (ang. *release channel*), zanim trafi w ostatecznej postaci do najszerszego grona odbiorców.

Rozwiązanie to nie niesie ze sobą tak wielkiego ryzyka, jak mogłoby się wydawać na pierwszy rzut oka. W rzeczywistości zanim produkt osiągnie fazę beta, musi przejść kilka innych etapów, na których sprawdzamy, czy jest wart przyznania mu etykiety „beta". Przeglądarka Chrome — nad tym projektem spędziłem pierwsze dwa lata pracy dla Google — była oferowana różnym grupom odbiorców w różnych kanałach dystrybucji, zanim zebraliśmy odpowiednio wiele opinii, by uznać, że jest produktem godnym zaufania. Na poszczególnych etapach rozwoju aplikacja jest dostępna w kilku różnych kanałach dystrybucji:

- **kanarkowym** — kanał ten wykorzystujemy do rozpowszechniania wczesnych, zmieniających się niemal codziennie wersji oprogramowania, które prawdopodobnie nie nadają się jeszcze do zaprezentowania szerszemu gronu odbiorców. Te „wypusty" odgrywają taką samą rolę jak kanarek w kopalni — jeśli aktualna wersja nie przetrwa fazy testów, stanowi to sygnał, że prace nad nią przebiegały zbyt chaotycznie i koniecznie trzeba zastanowić się nad ich kształtem. Wersje oferowane w kanale kanarkowym są przeznaczone wyłącznie dla bardzo odpornych psychicznie użytkowników, którzy są zainteresowani przede wszystkim prowadzeniem testów. Z pewnością nie należy proponować ich ludziom potrzebującym działającego w pełni programu. Ogólnie z kanału tego korzystają wyłącznie inżynierowie (programiści i testerzy) oraz kierownicy nadzorujący tworzenie danej aplikacji;

Uwaga

Zespół odpowiedzialny za rozwijanie systemu Android poszedł o krok dalej. Niemal wszyscy jego członkowie korzystają z wypuszczanych codziennie nowych wersji systemu. Chodzi o to, że jeśli sami nie będą mogli zadzwonić do domu, nie będą tak ochoczo przechodzić do porządku dziennego nad błędami w kodzie.

- **deweloperskim** — z kanału tego korzystają na co dzień programiści. Za jego pośrednictwem udostępnia się zazwyczaj wersje będące wynikiem prac z całego tygodnia, które sprawdziły się już w niektórych obszarach i przeszły niektóre z testów (do tego tematu wrócimy w dalszych rozdziałach). Wszyscy pracujący nad danym produktem mają *obowiązek* pobierać aplikacje udostępniane tą drogą, korzystać z nich w pracy i przeprowadzać testy. Jeśli wersja deweloperska aplikacji nie sprawdzi się w codziennych zdaniach, wraca do kanału kanarkowego. Tego rodzaju wydarzenia nie dają powodów do radości i zazwyczaj prowadzą do wzmożonych badań mających pozwolić raz jeszcze ocenić całość projektu;
- **testowym** — tym kanałem rozpowszechnia się zazwyczaj wersję, którą można by nazwać mianem zwycięzcy miesiąca, czyli taką, która przejdzie pozytywnie większość prowadzonych na tym etapie testów i cieszy się największym uznaniem w oczach jej autorów. Z kanału testowego korzystają wewnętrzni dogfooderzy, a prezentowana im wersja aplikacji jest zazwyczaj silnym kandydatem na pozycję wersji beta. Na pewnym etapie prac wersja testowa staje się na tyle stabilna, że można używać jej bezpiecznie wewnątrz firmy, a z czasem udostępniać także wybranym współpracownikom zewnętrznym i partnerom, którzy mogliby skorzystać na wcześniejszym poznaniu nowego towaru;
- **beta** lub **RC** — do tego etapu docierają stabilne wersje testowe, które zostały sprawdzone już w wewnętrznym użyciu firmowym i zdołały pokonać wszystkie poprzeczki testowe stawiane przez członków zespołu deweloperskiego. To one jako pierwsze trafiają do szerszego grona odbiorców.

Stabilny rozwój — od raczkowania, przez chód, aż do biegu — pozwala nam przeprowadzać testy już w najwcześniejszych wersjach aplikacji i zbierać uwagi nie tylko z prowadzonych codziennie w każdym z kanałów automatycznych testów programowych, lecz także od ludzi, którzy mieli styczność z produktem.

Rodzaje testów

Google nie stosuje typowego podziału testów na testy kodowania, integracji i systemowe. Zamiast takiego rozróżnienia wprowadzamy własną nomenklaturę — testy *małe*, *średnie* i *duże* (nie mylmy tych pojęć ze stosowanymi przez lotną społeczność przybliżonymi rozmiarami ubrań). Stosowane nazwy mają oddawać zasięg prowadzonych badań. Za pomocą małych testów sprawdzamy niewielkie

wycinki kodu i tak dalej. Każda ze wspomnianych wcześniej grup inżynierów może zajmować się prowadzeniem dowolnego rodzaju testów; testy te mogą mieć charakter tak automatyczny, jak i ręczny.

Zamiast przeprowadzać osobno testy kodowania, integracji i systemowe, Google woli stosować *małe*, *średnie* bądź *duże* testy — w zależności od tego, jaki zakres aplikacji jest sprawdzany za ich pomocą.

Małe testy obejmują przede wszystkim (choć nie tylko) weryfikację automatyczną, której celem jest sprawdzenie kodu zawartego wewnątrz jednej funkcji czy w danym module. W czasie ich trwania bada się przede wszystkim sposób działania kodu, szuka błędów zapisu danych, określa warunki ich występowania i wyszukuje błędy logiczne typu *off-by-one*, czyli pomyłki polegające na przesunięciu o jeden wartości dyskretnych, bardzo często o charakterze brzegowym. Małe testy nie pochłaniają wiele czasu — zazwyczaj przeprowadza się je w kilka sekund. Odpowiednie funkcje przygotowują IO, rzadziej ITO, a IT prawie nigdy nie mają z nimi do czynienia. Małe testy prowadzi się w środowiskach próbnych za pomocą tak zwanych mocków i fake'ów. (Mock i fake to odmiany obiektów typu stub, czyli obiekty zastępujące rzeczywiste funkcje. Przechowuje się w nich funkcje, które być może w ogóle nie zostaną wykorzystane w ostatecznej wersji aplikacji, są zbyt niedopracowane, by można było ich użyć, czy zbyt skomplikowane, by określić warunki, w jakich mogą wystąpić w nich błędy. W dalszych rozdziałach wrócę jeszcze do tego tematu). Inżynierowie testujący nie piszą wprawdzie zbyt często małych testów, za to zdarza się im uruchamiać je, by określić przyczyny występowania konkretnego błędu. Małe testy mają za zadanie uzyskać odpowiedź na pytanie: „Czy ten fragment kodu działa tak, jak powinien?".

Średnie testy przeprowadza się najczęściej automatycznie, by sprawdzić za ich pomocą przynajmniej dwie współdziałające funkcje aplikacji. Na tym etapie sprawdza się przede wszystkim relacje funkcji wywołujących się wzajemnie albo oddziałujących w jakiś sposób na siebie. Funkcje tego rodzaju nazywamy *najbliższymi sąsiadami*. Inżynierowie z grupy ITO nadzorują przygotowanie tych testów jeszcze we wczesnej fazie pracy nad aplikacją, równolegle do przygotowywania poszczególnych funkcji. Za przygotowanie ich od strony kodu, sprawdzenie poprawności i przeprowadzenie testów odpowiadają zazwyczaj IO. W razie gdyby średni test nie dał się uruchomić lub dawał błędne wyniki, programista może sam wprowadzić w nim poprawki. W późniejszej fazie prac nad aplikacją średnie testy wykonują inżynierowie testujący — czasami ręcznie (jeśli automatyzacja procesu byłaby zbyt trudna lub zbyt kosztowna), czasami zaś automatycznie. Średnie testy pozwalają szukać odpowiedzi na pytanie: „Czy ten zestaw sąsiadujących ze sobą funkcji działa w sposób, w jaki powinien?".

Duże testy obejmują sprawdzenie współdziałania przynajmniej trzech (a zazwyczaj większej liczby) funkcji aplikacji i zawierają scenariusze faktycznego użytkowania. Przeprowadzenie ich zajmuje zazwyczaj co najmniej kilka godzin. Choć przy okazji

sprawdza się ogólny stopień zintegrowania poszczególnych funkcji, to najważniejszym ich aspektem jest badanie zwracanych w czasie ich trwania wyników, na podstawie których ocenia się, czy program spełni oczekiwania użytkownika. W pracach nad przygotowaniem dużych testów biorą udział wszystkie grupy inżynierów, a sam test może przebiegać w dowolny sposób — od pełnego zautomatyzowania procesów do prowadzenia ręcznie testów badawczych. Przeprowadzając duże testy, staramy się odpowiedzieć na pytanie: „Czy aplikacja działa w sposób, jakiego oczekiwałby użytkownik, i czy wyniki jej pracy są zadowalające?”. Do tej grupy testów zalicza się między innymi przekrojowe scenariusze badające wyniki pracy pełnych wersji oprogramowania i usług.

Uwaga

Małe testy mają za zadanie zweryfikować poprawność działania niewielkich fragmentów kodu uruchamianego w sztucznie przygotowanym środowisku. **Średnie testy** pozwalają sprawdzić działanie wielu współdziałających ze sobą modułów w warunkach sztucznie przygotowanych, ale także w faktycznym środowisku działania aplikacji. **Duże testy** pozwalają badać dowolną liczbę modułów uruchamianych w dedykowanym im środowisku i z wykorzystaniem autentycznych zasobów.

Sama nomenklatura nie jest istotna. Nie musisz nazywać testów małymi, średnimi i dużymi, jeśli nie masz ochoty — grunt, żeby zastosowane nazwy nie wprowadzały nikogo w błąd[7]. Najważniejsze, że testerzy w Google mają język, w którym mogą się porozumiewać i uzgadniać zakres prowadzonych prac. A jeśli któremuś bardziej rzutkiemu zamarzyłoby się przeprowadzenie testów jeszcze wyższego poziomu, w związku z czym wprowadziłby nazwę *olbrzymie testy*, każdy inny pracownik firmy domyśliłby się od razu, że chodzi o sprawdzenie działania całego systemu, każdej jego funkcji, i że należy na to zarezerwować odpowiednio dużo czasu. Żadne dodatkowe wyjaśnienia nie są już konieczne[8].

Wytyczne dotyczące elementów poddawanych testom i dynamiki procesu weryfikacji zmieniają się z projektu na projekt. Google preferuje politykę wprowadzania na rynek częstych aktualizacji, co pozwala szybciej prezentować kolejne wersje

[7] Nazewnictwo stosowane w Google zrodziło się z autentycznej potrzeby. Okazało się, że należy ustandaryzować terminologię stosowaną przez testerów, którzy pracowali przecież wcześniej w innych firmach, gdzie posługiwano się zupełnie innymi nazwami — czasami były to testy dymne, czasami testy BVT, czasem integracyjne. Terminom tym przypisywano różne, niejednokrotnie przeciwstawne znaczenia, uznano zatem, że Google potrzebuje własnych określeń.

[8] I rzeczywiście idea olbrzymich testów trafiła do oficjalnej dokumentacji. W infrastrukturze Google funkcjonują na co dzień pojęcia testów małych, średnich i tak dalej — w ten sposób definiuje się kolejkę wykonywania zadań w czasie trwania testów automatycznych. Więcej szczegółów znajdziesz w rozdziale poświęconym zadaniom stawianym przed inżynierami do spraw testowania.

użytkownikom i szybciej zbierać ich odzew. W ten sposób przyspieszamy prace nad kolejnymi zmianami. W Google staramy się rozwijać tylko te aplikacje i usługi, które przypadły użytkownikom do gustu, i wprowadzać nowe funkcje tak szybko, jak jest to możliwe, by odbiorca nie musiał długo na nie czekać. Jednocześnie jesteśmy w stanie ograniczyć liczbę zbędnych rozwiązań na bardzo wczesnym etapie. Oczywiście tego rodzaju model działania wymaga wczesnego wydania aplikacji w ręce użytkowników i zewnętrznych programistów, ale właśnie dzięki niemu mamy pewność, że oferujemy ludziom dokładnie to, czego potrzebują.

Wreszcie trzeba stwierdzić wyraźnie, że w połączeniu testów automatycznych z prowadzonymi ręcznie te pierwsze stoją na zdecydowanie uprzywilejowanej pozycji. Jeśli zagadnienie da się zautomatyzować, a sam problem nie wymaga odwoływania się do ludzkiej intuicji i zdolności umysłowych, należy bezwzględnie stosować rozwiązania maszynowe. Testy ręczne warto stosować tylko tam, gdzie osąd człowieka jest absolutnie niezbędny, na przykład podczas podejmowania decyzji dotyczących estetyki interfejsu użytkownika czy określania, które z danych mogą naruszać prawo do prywatności.

We wszystkich trzech opisanych wcześniej metodach testowania w połączeniu testów automatycznych i ręcznych te pierwsze sprawdzają się znacznie lepiej. Jeśli zatem zagadnienie da się zautomatyzować, a sam problem nie wymaga odwoływania się do ludzkiej intuicji i zdolności umysłowych, należy bezwzględnie stosować rozwiązania maszynowe.

To jedna strona medalu. Jednocześnie Google przeprowadza mnóstwo testów ręcznych, zarówno skryptowych, jak i eksploracyjnych, ale nawet one przebiegają pod czujnym okiem zautomatyzowanych procedur. Techniki nagrywania pozwalają tworzyć testy automatyczne na podstawie zapisów z przeprowadzanej wcześniej ręcznej weryfikacji — każde przesunięcie kursora i każde kliknięcie jest rejestrowane, by w kolejnych wersjach aplikacji móc wykorzystać te nagrania i dzięki temu zminimalizować regresję prac nad programem. Takie rozwiązanie pozwala testerom pracującym własnoręcznie nad badaniem aplikacji zająć się nowymi problemami. Automatyzujemy też procedurę przesyłania raportów i organizowania wykonywanych ręcznie zadań testowych[9]. Jeżeli przykładowo funkcja nie przejdzie automatycznych testów poprawnie, system wskaże ostatnie z wprowadzonych w kodzie zmian, typując tym samym najbardziej prawdopodobnego sprawcę nieszczęścia, wyśle wiadomość do jej autora i sam wprowadzi do dziennika odpowiednią informację o wystąpieniu błędu. Nieustannie podejmowane wysiłki, mające na celu wprowadzić automatyzację również w ostatnim odcinku ludzkiego umysłu, stanowią swojego rodzaju specyfikację narzędzi testowych nowej generacji, jakie powstają w Google.

[9] Więcej na temat mechanizmów rejestrujących i automatyzacji testów ręcznych znajdziesz w rozdziałach poświęconych pracy inżynierów testujących.

Skorowidz

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

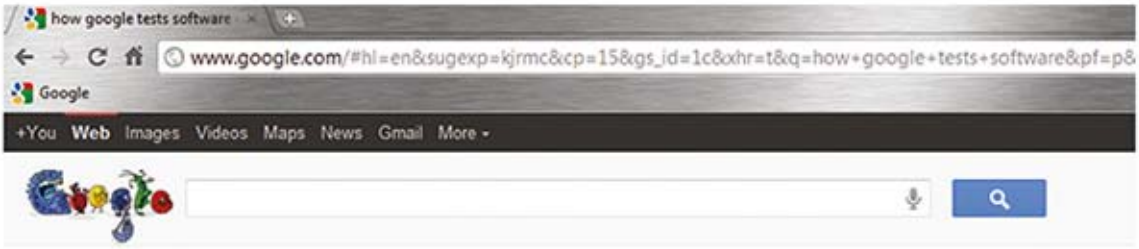
V

W

Y

Z

Ż



Sprawdź, jak testują najlepsi!

Oprogramowanie firmy Google to miliony linii kodu źródłowego, dziesiątki wersji językowych, różne systemy operacyjne, przeglądarki i preferencje użytkownika. Jak przy takich wymogach dostarczyć klientom produkt najwyższej jakości? Tu mogą pomóc tylko testy automatyczne. Dzięki nim codziennie bez trudu można uruchomić miliony testów! Google opanowa tę sztukę do mistrzostwa. Warto uczyć się od najlepszych!

Dzięki tej książce dowiesz się, jak zorganizować proces testowania, tak aby był elastyczny, skuteczny i spełniał Twoje oczekiwania. Poznasz rolę inżyniera do spraw testowania oprogramowania, kierownika zespołów inżynierskich oraz inżyniera testującego. Zobaczysz, na jakie problemy natykają się oni każdego dnia oraz jak sobie z nimi radzą. Ponadto nauczysz się oceniać ryzyko, dokumentować proces testowania i raportować błędy. Książka ta jest obowiązkową lekturą dla wszystkich osób, które doskonalą swoje umiejętności programistyczne i chcą polepszyć jakość dostarczanego oprogramowania.

Dzięki tej książce:

- poznasz najlepsze metody zapewniania jakości oprogramowania
- nauczysz się planować i przeprowadzać testy
- poprawnie ocenisz ryzyko
- udoskonalisz proces wytwarzania oprogramowania!

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu

ISBN 978-83-246-8656-8
9 788324 686568
cena: 67,00 zł

ALWAYS LEARNING

PEARSON