Paweł Baszuro

U MNIE DZIAŁA

JĘZYK BRANŻY IT

Helion

Redaktor prowadzący: Małgorzata Kulik

Grafika na okładce została wykorzystana za zgodą Shutterstock.com

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/umnied
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-5613-9

Printed in Poland.

Spis treści

Testowanie

Ćmy i motyle to owady bardzo podobne do siebie. Polowanie na motyle kojarzymy z chwytaniem ich w siatkę na wiosennej polanie. Dla kontrastu łapanie ćmy wiążemy z mniej finezyjnym wymachiwaniem klapkiem. Ćmy mają to do siebie, że czasem, wiedzione świetlnym instynktem, podejmują się samobójczych misji, celując w rozgrzane przedmioty. Jednak dzięki temu pewnego dnia 1947 roku programistka Grace Hopper odnotowała wadliwe działanie komputera Mark II. Wadliwe działanie było spowodowane przez znalezioną w elektromechanicznych podzespołach ćmę. Hopper oraz członkowie

jej zespołu zaczęli używać terminu „bug" (pluskwa, owad) do opisu niewłaściwego działania komputera. W latach czterdziestych dwudziestego wieku na całym świecie było zaledwie kilka takich urządzeń, a informatyka i programowanie dopiero raczkowały. Podobnie było z językiem opisującym różne zjawiska związane z komputerami, stąd termin „bug" został spopularyzowany do opisu dowolnego, nieoczekiwanego zachowania komputera (niekoniecznie związanego z insektami). Jak wskazują historycy domeny[11], sam termin został po raz pierwszy użyty w 1873 roku przez Thomasa Edisona przy okazji pracy nad urządzeniem do przesyłania telegramów, jednak „bug" na stałe zagościł w żargonie programistycznym do opisu błędów.

Błędy w oprogramowaniu mają różne pochodzenie, począwszy od niedociągnięć programistów, przez nieprzewidziane sytuacje, po błędy wynikające z wymagań. Liczba błędów w oprogramowaniu jest skorelowana z szeroko rozumianą jakością oprogramowania.

Niedociągnięcia programistów wynikają z braku wiedzy lub doświadczenia konkretnego programisty, jego pośpiechu

[11] *Did You Know? Edison Coined the Term „Bug", Bugs have plagued technologists for centuries*, Alexander B. Magoun, Paul Israel, IEEE, 23 sierpnia 2013, w języku angielskim dostępny pod adresem *http://theinstitute.ieee.org/tech-history/technology-history/did-you-know-edison-coined-the-term-bug*.

lub nieuwagi. Takie błędy są zazwyczaj relatywnie łatwe do naprawienia lub wyłapania w trakcie profesjonalnego procesu wytwarzania (faza testów).

Nieprzewidziane sytuacje, takie jak nieoczekiwane działanie sprzętu (np. brak odpowiedzi lub długotrwałe przetwarzanie), nieoczekiwany ciąg zdarzeń użytkownika (np. wybieranie opcji na ekranie w kolejności innej niż zakładana) lub kombinacja obu, są trudniejsze do naprawienia. Zanim programista będzie mógł skutecznie naprawić błąd, musi najpierw zidentyfikować przyczynę i powtórzyć ciąg zdarzeń wiodących do wystąpienia błędu. Dlatego istotne jest dokładne udokumentowanie zgłoszenia błędu lub defektu. Na poziomie technicznym analizę zachowania komputera podczas pracy programu nazywamy debugowaniem (ang. *debugging*, inne polskie tłumaczenie to odpluskwianie). Programiści, pracując nad złożonymi problemami, często prowadzą analizy typu post mortem, czyli analizy po fakcie wystąpienia buga. Aby skutecznie przeprowadzić tego typu analizę, potrzebne są różne artefakty techniczne, takie jak pliki logów, zrzuty pamięci i pliki symboli.

Pliki logów to ciąg notowań różnych akcji realizowanych przez działający program. Ze względu na poziom skomplikowania współcześnie używanego oprogramowania nie jest możliwe odnotowanie każdej potencjalnej instrukcji (względy wydajności; czasem także ze względu na dostępną powierzchnię

dyskową), stąd domyślne logowanie zdarzeń jest wyłączone. Niektóre aplikacje po wykryciu błędu automatycznie aktywują logowanie, w innych przypadkach użytkownik może zostać poproszony o zmianę konfiguracji programu. Zaawansowany użytkownik może być także poproszony o zrobienie i udostępnienie zrzutu pamięci programu.

Zrzut pamięci (ang. *memory dump*) zawiera stan pracy programu wraz z wczytanymi bibliotekami, stanem używanych urządzeń oraz aktualnie przetwarzanymi danymi użytkownika. Zrobienie zrzutu pamięci działającego programu umożliwione jest przez system operacyjny (powstały plik może mieć duży rozmiar). Za pomocą narzędzi programistycznych można otworzyć plik zrzutu pamięci i przeanalizować ciąg aktualnie przetwarzanych instrukcji oraz zawartości pamięci. Pliki symboli pozwalają programiście połączyć zrzut pamięci z kodem źródłowym programu (wiele instrukcji wykonywanego programu przekłada się na jedną instrukcję w kodzie źródłowym) .

Najtrudniejszymi błędami są błędy wynikające z wadliwych, nieprecyzyjnych lub sprzecznych wymagań. Takie błędy wymagają ponownej analizy wymagań, poprawienia ich oraz zaprojektowania, zaimplementowania i przetestowania nowego rozwiązania. Ze względu na mnogość zadań i czas poświęcony na dostarczenie poprawionej wersji jest to najbardziej kosztowny bug.

Zgłoszenia błędów

„Okrągłe" liczby w świecie IT, jak 64 czy 1024, mogą wydawać się trudne do zapamiętania dla przeciętnego użytkownika. Są to jednak liczby nieprzypadkowe, to potęgi liczby dwa. Liczba dwa jest związana z dwoma poziomami napięcia — napięcie jest (poziom jeden) lub go nie ma (poziom zero). Ciąg zmian poziomu napięcia tworzy ciąg zer i jedynek. Jedynka na kolejnej pozycji oznacza potęgę dwójki. I tak liczba 1000 w zapisie binarnym oznacza liczbę 8. W pamięci komputera wszystkie dane są reprezentowane właśnie w postaci binarnej (kod binarny). Ze względu na łatwiejsze zarządzanie tymi ciągami liczb bity przyjęło się grupować po osiem i nazywać bajtami. Osiem bitów to 256 możliwości. Profesor uniwersytetu Stanforda Donald Knuth zwykł wystawiać czeki na okrągłą liczbę 2,56 dolara dla każdej osoby (okrągłe 256 centów), która znajdzie błąd lub będzie miała znaczący wkład w jego pracę[12]. Donald Knuth być może zapoczątkował program nagradzania za wyszukiwanie błędów w oprogramowaniu (ang. *bug bounty program*). Tego typu program polega na wynagradzaniu każdego, kto znajdzie bug w oprogramowaniu, w podzięce za zgłoszenie. Zazwyczaj im większa firma, im groźniejsze ryzyko

[12] Wpis w Wikipedii w języku angielskim dostępny pod adresem *https://en.wikipedia.org/wiki/Knuth_reward_check*.

związane z błędem, tym wyższa wartość materialna wynagrodzenia. Firma Facebook, stojąca za największą siecią społecznościową (w momencie pisania tej książki serwis Facebook miał ponad dwa miliardy użytkowników), w podzięce za zgłoszenie błędu związanego z bezpieczeństwem zwykła rozsyłać czarne karty kredytowe[13]. Zaś firma Microsoft za pomocą programu Windows Insider[14] zbiera opinie (w tym także zgłoszenia o błędach) na temat kolejnych wersji systemu Windows, zanim nowe wydania staną się publicznie dostępne dla szerokiego grona odbiorców. Tym samym każdy uczestnik programu może wpłynąć na kształt jednego z najpopularniejszych systemów operacyjnych.

Warto zatem wiedzieć, jak napisać dobre zgłoszenie błędu. Bug wiąże się z konkretnym oprogramowaniem, akcją i reakcją systemu. Przygotowując zgłoszenie, należy sprawdzić wersję oprogramowania (lub jego komponentu), które podlega zgłoszeniu. Jeśli oprogramowanie działa w wielu systemach i na wielu platformach, także je warto scharakteryzować (np. jaki

[13] Facebook hands out White Hat debit cards to hackers, ELINOR MILLS, CNET, 31 grudnia 2011, w języku angielskim dostępny pod adresem *https://www.cnet.com/news/facebook-hands-out-white-hat-debit-cards-to-hackers/*.

[14] Strona Microsoft Windows Insider Program, dostępna pod adresem: *https://insider.windows.com/pl-pl/*.

system operacyjny, w jakiej wersji, jaka przeglądarka internetowa i w jakiej wersji). W zależności od typu zgłoszenia charakterystyka urządzenia może być istotna (np. nazwa i model telefonu komórkowego). Im więcej tego typu szczegółów umieścimy, tym szybciej programiści będą mogli zbadać problem.

W ramach zgłoszenia należy precyzyjnie przedstawić ciąg kolejnych kroków, które wiodły do odnalezienia błędu. Taki ciąg można spisać w punktach, np. „Wybranie z menu *Plik* opcji *Otwórz*; w nowym oknie zaznaczenie pliku; wybranie z akcji *Anuluj*". Powinniśmy również napisać, czego oczekiwaliśmy w tym momencie, np. „oprogramowanie otwiera zaznaczony plik". Warto także wyartykułować nasze oczekiwania po takiej akcji, np. „oprogramowanie zamknie okno oraz nie otworzy zaznaczonego pliku". Dobrą praktyką jest sprawdzenie przed zgłoszeniem, czy błąd miał charakter jednostkowy, czy może powtarza się za każdym razem (także po restarcie urządzenia). Inne „znaleziska" warto odnotować w komentarzach do zgłoszenia.

Do zarządzania stanem zgłoszeń często używa się dedykowanych systemów zarządzania zgłoszeniami (ang. *bug tracking system*), np. JIRA lub Redmine.

Rodzaje testów

Ludzkość przez tysiąclecia traktowała upuszczanie krwi jako ważny element terapii różnych chorób. Nietrudno sobie wyobrazić kapłana starożytnej religii, który źródła gorszego stanu zdrowia upatruje w „złej" krwi, więc należy się jej pozbyć. Tak postawiona hipoteza wymagała natychmiastowej weryfikacji. Uzdrowiony chory utwierdzał kapłana w pozytywnym działaniu terapii (test pozytywny). Co ciekawe, chory bez jakiejkolwiek innej kuracji zapewne nie odczuwał poprawy, co także mogło służyć do potwierdzenia tej tezy (test negatywny). Takie obserwacje zapisały upuszczanie krwi jako istotny element terapii medycznej na kolejne stulecia. W XIX wieku zaczęto weryfikować skuteczność tej metody i dziś jest ona stosowana w nielicznych schorzeniach. Przetestowanie tego rozwiązania zajęło jednak kilka tysięcy lat. Oczekiwaniem względem przedstawionej metody była poprawa stanu zdrowia chorego, co udowadniało prawdziwość tej (pozytywnej) hipotezy. Nieprecyzyjna ocena stanu zdrowia przed i po zastosowaniu metody leczniczej pozwalała na zakończenie jej testów z sukcesem.

Powyższy przykład pozwala na wyprowadzenie bardziej ogólnych wskazówek na temat testowania (w tym także oprogramowania). Po pierwsze w testach należy być precyzyjnym w określaniu stanu wejściowego (w testach „aranżujemy" stan wejściowy, ang. *arrange*), podejmowanych kroków (ang. *act*)

i stanu oczekiwanego (ang. *assert*). Po drugie należy zweryfikować nie tylko pozytywne scenariusze, ale również negatywne (np. jak powyższa metoda zadziała na zdrowego człowieka?) i przypadki skrajne (np. jak pacjent będący na granicy progu wyleczenia innymi metodami zachowa się pod wpływem wyżej wymienionej metody?). Weryfikując wyniki testu, należy skategoryzować wyniki — co to znaczy, że test zakończył się sukcesem? Kiedy mówimy o porażce?

Sukcesem (ang. *test success* lub *test pass*) jest każdy wynik testu, który wskazuje, że oprogramowanie zachowuje się zgodnie z oczekiwaniami (zarówno testy pozytywne, jak i negatywne mogą, a wręcz powinny kończyć się sukcesem). Porażką (ang. *test fail*) jest każdy test, w wyniku którego otrzymany stan jest inny od oczekiwanego.

Mówiąc o precyzji, należy jasno określić metodę testowania oraz jej zakres (najlepiej zanim zlecimy zespołowi technicznemu pracę nad oprogramowaniem). Wyróżnia się testy jednostkowe, komponentowe, integracyjne oraz systemowe. Testy jednostkowe polegają na testowaniu bliżej niezdefiniowanej „jednostki". Tutaj pojawia się pytanie, czym jest ta mityczna „jednostka"? Jednostka to najmniejszy moduł w oprogramowaniu, który da się przetestować. W praktyce oznacza to zwykle pojedynczą linijkę kodu i odpowiedzenie na pytanie, czy aby na pewno funkcjonuje ona prawidłowo i czy jest to

udowodnione (pokryte) testami. Sprawy nieco komplikują się, kiedy testujemy decyzję. W takim wypadku należy przetestować wszystkie możliwe rozgałęzienia danej decyzji (ang. *branch testing*; np. badając warunek, czy liczba jest podzielna przez dwa, powinniśmy przetestować zarówno przypadki, kiedy testowana liczba jest liczbą parzystą, jak i nieparzystą). Często spotykamy się z sytuacją, kiedy prowadzone są metryki testów, w których gromadzone są informacje na temat stosunku przetestowanych linijek kodu do liczby wszystkich linijek kodu, stosunku przetestowanych rozgałęzień do wszystkich rozgałęzień w kodzie, stosunku przetestowanych plików, bytów do wszystkich plików, bytów w kodzie itd. Takie testy realizowane są przez programistów. Jedną z miar oceny jakości oprogramowania jest procent pokrycia kodu testami.

Przechodząc do testów komponentu, zazwyczaj testujemy zespół lub ciąg linijek kodu, które składają się na większy, samodzielny obszar realizujący określoną funkcjonalność. Testy komponentu zazwyczaj nie mają aż tak dużej granulacji jak testy jednostkowe, pokrywają przepływ danych i sterowania przez komponent, od wejścia do wyjścia danych. W praktyce oznacza to testowanie komponentu w odizolowaniu od pozostałych (np. testowanie komponentu generowania raportu w odizolowaniu od komponentu wprowadzania danych). Istnieją przypadki, kiedy testy komponentu nie mogą być

przeprowadzone samodzielnie przez użytkownika i wymagana jest pomoc zespołu programistów. Takie wsparcie może być potrzebne w budowaniu zaślepek dla komponentów wchodzących w interakcję z testowanym komponentem (ang. *stub*) albo symulowaniu zachowania tychże komponentów (ang. *mock*). Innym przypadkiem testów, które także wymagają nakładu prac zespołu technicznego, są testy integracyjne.

Testy integracyjne mają na celu sprawdzenie, czy komponenty poprawnie współpracują ze sobą. Często testuje się komponenty parami. Podczas testów integracyjnych sprawdza się, czy dane z jednego komponentu trafiają do drugiego w poprawnej i niezniekształconej postaci oraz w dobrej kolejności (np. czy moduł generowania raportu przekaże poprawny raport do modułu wydruku raportu).

Testy systemowe z kolei polegają na testowaniu systemu całościowo. Testy systemowe mogą być przekrojowe i przechodzić przez wszystkie komponenty składające się na system (np. definiujemy dwie pozycje, jakie mają być wprowadzone do systemu; wprowadzamy je do systemu; wybieramy opcję generowania raportu dla nich; następnie wybieramy opcję wydruku wygenerowanego raportu i weryfikujemy, czy wydruk zawiera wprowadzone wcześniej pozycje).

Kiedy podczas testów realizowane są typowe scenariusze użycia oprogramowania, naśladujące zachowanie realnego

użytkownika, to wtedy mówimy o testach akceptacyjnych (ang. *user acceptance test*, UAT).

Oprogramowanie możemy testować manualnie albo ułatwić sobie pracę i testować automatycznie. Testy manualne (innymi słowy ręczne) wymagają interwencji użytkownika (potocznie „przeklikania się" przez system) i zazwyczaj są czasochłonne. Testy manualne pozwalają na wygenerowanie przypadkowych zdarzeń w systemie, które mogą prowadzić do znalezienia nieoczywistych błędów. Takie testowanie nazywamy testowaniem eksploracyjnym. Częste testowanie systemu wymaga dużego nakładu czasu i środków na przetestowanie całości systemu. Automatyzacja znanych i typowych testów przyczyni się do minimalizacji kosztów. W takich przypadkach to komputer przetestuje oprogramowanie (podejmując takie same akcje jak człowiek siedzący przed komputerem) oraz zdecyduje, czy wynik zachowania oprogramowania jest zgodny z oczekiwanym (wprowadzonym wcześniej przez człowieka do testu automatycznego).

Zbiór wszystkich testów, które są powtarzane przy każdej sesji testowania (np. z okazji wydania nowej wersji), nazywa się testami regresyjnymi. Na testy regresyjne mogą składać się zarówno testy manualne, jak i testy automatyczne. Celem tych testów jest zweryfikowanie, czy podstawowe funkcje oprogramowania działają w niezmienionej i oczekiwanej formie.

Wiele zespołów technicznych realizuje testy regresyjne przy każdej większej zmianie dowolnego z elementów oprogramowania (np. zmian w kodzie źródłowym, zmian w zbiorze używanych bibliotek lub znaczących zmian w konfiguracji komponentu).

Bez względu na to, jakie rodzaje testów zostały przeprowadzone, należy każdorazowo zgromadzić artefakty (dokumenty, raporty, zrzuty ekranu i wideo) potwierdzające fakt przeprowadzenia testów i ich rezultaty. Takie artefakty nazywa się ewidencją testów (ang. *test evidence*).

Środowisko testowe

Woda wrze w temperaturze stu, a zamarza poniżej zera stopni Celsjusza. Na podstawie tych faktów można byłoby przetestować, czy termometr pokazuje właściwą temperaturę. Jednak kiedy z tym samym termometrem udałoby nam się wspiąć na wysoką górę (np. Mount Everest), wtedy wskazałby znacząco inną temperaturę dla tych samych zjawisk. Zmiana środowiska w tym wypadku powoduje inny punkt odniesienia (względem przemian termodynamicznych) i musi to być uwzględnione w testach. W idealnym przypadku test powinien odbywać się w takim samym środowisku (a w opisanym przypadku pod takim samym ciśnieniem), aby uzyskać takie same wyniki.

Nie inaczej jest z testami oprogramowania. Środowisko, w którym odbywają się testy, ma równie istotne znaczenie.

Każdy z wyżej wymienionych rodzajów testów realizowany jest (w najbardziej optymistycznym scenariuszu) w osobnych środowiskach testowych. Testy jednostkowe zazwyczaj wykonywane są w środowisku programistycznym (ang. *development environment*, w skrócie *dev environment*), testy komponentów w środowisku zapewnienia jakości (ang. *quality assurance environment*, *QA environment*), testy integracyjne w środowisku integracyjnym (ang. *integration environment*, *INT environment*), zaś testy całego systemu z perspektywy użytkownika odbywają się w środowisku akceptacyjnym (ang. *User Acceptance Test environment*, *UAT environment*). Środowisko nietestowe i z realnymi danymi nazywamy produkcją (ang. *Production environment*, *PROD environment*). Podczas wytwarzania oprogramowania czasem używa się dodatkowych środowisk albo całkowicie innej terminologii nazewniczej. Istnieją także przypadki ograniczenia liczby środowisk testowych (np. ze względu na koszty zarządzania lub zakupu licencji). W ostatnich latach istnieje trend automatyzacji procesu tworzenia, zarządzania i konfigurowania różnych środowisk, począwszy od automatyzacji opartej na prostych skryptach systemowych, po rozwiązania oparte na wirtualnych maszynach (np. Docker i zarządzanie konfiguracją za pomocą narzędzia o nazwie Kubernetes).

Skorowidz

A

B

C

D

E

F

K

L

M

N

O

P

Q

R

S

Ś

T

U

V

W

X

Z


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

NIE ROZUMIESZ JĘZYKA BRANŻY IT?

SIĘGNIJ PO TĘ KSIĄŻKĘ!

- **OPANUJ SŁOWNIK INFORMATYKÓW**
- **POZNAJ CYKL ŻYCIA OPROGRAMOWANIA**
- **NAUCZ SIĘ KOMUNIKACJI Z BRANŻĄ IT**

Kto choć raz miał do czynienia z informatykami, wie, że rozmowy z nimi często przypominają kontakty z przybyszami z innej galaktyki. Posługują się specjalistycznym żargonem, są mocno skupieni na aspektach technicznych, bywa, że się niecierpliwią, gdy otoczenie nie nadąża za ich tokiem rozumowania. Jeśli rozmówca nie jest zorientowany w branży IT, zwykle trudno mu znaleźć wspólny język z przedstawicielami środowiska i może mieć nie lada kłopot, gdy będzie musiał coś z nimi załatwić.

TA KSIĄŻKA BĘDZIE PRAWDZIWYM OBJAWIENIEM DLA WSZYSTKICH, KTÓRZY WSPÓŁPRACUJĄ Z OSOBAMI ZATRUDNIONYMI W IT. Jej celem jest zaprezentowanie podstawowych terminów używanych w tej dziedzinie, przedstawienie procesu wytwarzania i utrzymania oprogramowania oraz wsparcie w zrozumieniu zagadnień i problemów, które dla informatyków stanowią codzienność. Lektura pomoże tym, którzy chcą poznać specyfikę branży lub są na różne sposoby zaangażowani w projekty IT.

- **TERMINOLOGIA UŻYWANA W ŚWIECIE INFORMATYCZNYM**
- **ETAPY WYTWARZANIA I WDRAŻANIA OPROGRAMOWANIA**
- **DEFINIOWANIE WYMAGAŃ I ZROZUMIENIE PROCESU**
- **KOMUNIKACJA Z PRZEDSTAWICIELAMI ŚRODOWISKA**
- **PRAKTYCZNE PRZYKŁADY Z ŻYCIA CODZIENNEGO**
- **OPANUJ INFORMATYCZNY ŻARGON W MGNIENIU OKA!**

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-5613-9
9 788328 356139
Cena: 34,90 zł

INFORMATYKA W NAJLEPSZYM WYDANIU