Testowanie użyteczności z przykładami

Testy użyteczności definiuje się jako metodę oceny produktu poprzez testowanie go na potencjalnych odbiorcach. Jest to również weryfikacja pewnych atrybutów niefunkcjonalnych oprogramowania w klasycznym ujęciu przedstawionym w sylabusie ISTQB.

30 lat temu Duńczyk Jakob Nielsen, pracujący w firmie Sun Microsystems, sformułował heurystyki dotyczące użyteczności, które można wykorzystać jako listę kontrolną, przydatną przy tego rodzaju testach. Odniosę się do tych reguł, nawiązując do poszczególnych, składających się na użyteczność aplikacji subcharakterystyk.

Łatwość użycia (operability)

Łatwość użycia określa stopień, w jakim oprogramowanie zapewnia użytkownikowi intuicyjne korzystanie z aplikacji. Podczas testowania tej cechy weryfikuje się m.in. spójność, w tym spójność interfejsu charakteryzującą się stosowaniem takiego samego stylu graficznego, układu menu, czcionek itp. w różnych modułach systemu oraz zgodność terminologii, czyli używanie analogicznych określeń w podobnych sytuacjach. Przyjrzyjmy się hipotetycznej sytuacji logowania, podczas której złamana została zasada integralności. Na tym samym ekranie tester wywołuje kolejno 3 komunikaty błędów, z których każdy wygląda zupełnie inaczej – ma inną czcionkę, kolor, a dodatkowo jest w obramowaniu prostokąta prostego lub w owalu.
Kolejną istotną kwestią zapewniającą intuicyjne korzystanie z aplikacji są jasne i zrozumiałe komunikaty, zarówno informacyjne, ostrzegawcze, jak i te o defektach. Przywołajmy sytuację, gdy np. próbujemy uruchomić materiał filmowy w technologii 360 stopni na platformie player.pl. Przeglądarka nie obsłuży tego żądania i na ekranie pojawi się komunikat: „Wystąpił błąd, skontaktuj się działem pomocy”. Czy nie lepiej byłoby przekazać bardziej szczegółową treść: „Materiał nie jest dostępny dla Twojej przeglądarki. Spróbuj skorzystać np. z Google Chrome czy Firefox i odtwórz materiał ponownie”? Tester powinien wykazywać się tu empatią i wczuwać w sytuację zdezorientowanego użytkownika. Kolejną istotną kwestią jest to, czy etykiety, opisy, ikony przycisków są zrozumiałe czy hiperłącza są prawidłowo opisane i widoczne. Wypełniałem kiedyś wniosek online, zaś aplikacja pozwalała mi przechodzić do poprzednich i następnych ekranów za pomocą przycisków. Jednak po najechaniu na przycisk, który był hiperłączem, kursor myszy nie zmieniał się na „rączkę”. Dodam, że dla innego, analogicznego wniosku, kursor zmieniał się prawidłowo. Zatem w tym przypadku nie było jasnej informacji, że przycisk jest linkiem. Innym ciekawym zagadnieniem z dziedziny łatwości użycia jest możliwość dostosowania oprogramowania do indywidualnych potrzeb użytkownika (ang. customization). Obecnie trudno sobie wyobrazić popularną aplikację, której nie moglibyśmy skonfigurować wedle własnych preferencji. Tak więc dodajemy kolejne przydatne rozszerzenia do Slacka czy przeglądarki Chrome, zmieniamy układ interfejsu lub motywy na Gmailu albo ustawiamy tryb nocny podczas szukania pracy na justjoin.it. Czy zwracamy uwagę na takie detale podczas naszych codziennych, rutynowych testów?

Łatwość nauki (learnability)

Jest to zdolność oprogramowania do wspierania użytkownika w procesie nauki jego użycia. Aplikacje o wysokiej łatwości nauki mogą być efektywnie używane bez wcześniejszego szkolenia. W

przeciwnym przypadku odbiorca może mieć wrażenie, że program używa nieznanych mu pojęć lub metod i odczuwa on brak niezbędnych informacji potrzebnych do używania oprogramowania. Warto sprawdzić, czy aplikacja posiada sekcję pomocy, tutotriale, podpowiedzi, tooltipy, które pomogą użytkownikowi nauczyć się z niej korzystać. Dobrym przykładem jest np. witryna podatki.gov.pl, która w przejrzysty i wyczerpujący sposób przeprowadza użytkownika przez proces obsługi swoich dokumentów podatkowych. Łatwość nauki wiąże się z inną cechą użyteczności – zrozumiałością, która mierzy stopień, w jakim odbiorca potrafi rozpoznać, czy dany system zaspokoi jego potrzeby. Charakterystyka ta opiera się na pierwszym wrażeniu, jakie produkt wywarł na użytkowniku oraz na dokumentacji użytkowej. Ta cecha użyteczności jest zwiększana przez dołączenie do programu wersji demonstracyjnej, przykładowych danych wejściowych (np. tablica z danymi demo w systemie JIRA) czy przewodników. Testowanie zrozumiałości może polegać na weryfikacji jakości oraz ilości pomocniczych materiałów dołączonych do oprogramowania.

Ochrona przed użytkownika błędami (user error protection)

Aplikacja powinna w możliwie jak największym stopniu zapobiegać potencjalnym błędom użytkownika. Zwiększa to produktywność, ponieważ dzięki ograniczeniu ryzyka popełniania pomyłek wzrasta szybkość i wydajność wykonania zadania. Ochrona przed błędami to stopień, w jakim system chroni przed popełnianiem pomyłek. Popularnymi rozwiązaniami tego typu są tzw. techniki poka-yoke, czyli metody przeciwdziałające powstawaniu defektów. Nie od dziś wiadomo, że lepiej zapobiegać niż leczyć. Wg Nielsena należy stosować, gdzie to możliwe, rozwijalne listy wyboru lub obiekty typu radio button, zamiast zmuszać do wpisywania wartości z klawiatury. Również wybór daty z kalendarza zminimalizuje ryzyko pomyłki, zwiększy szybkość, komfort oraz zapewni spójność zapisu danej w bazie. Jak jeszcze ochronić użytkownika przed popełnieniem błędu? Informując go zawczasu, jakie są reguły gry. Kiedy np. będzie on chciał dołączyć załączniki do formularza, dowie się wcześniej, jakie formaty są obsługiwane. Nie doda 5 plików, tylko 2, jeśli uprzedzimy go o maksymalnej dopuszczalnej wartości. Dla przykładu Twitter podaje ściśle określony limit znaków dla tweetów i ostrzega, zanim ten limit zostanie przekroczony, z pozostałą liczbą znaków. Wspomniany już były pracownik Sun Microsystems podkreślał, że użytkownik powinien być na bieżąco informowany o tym, w jakim miejscu systemu się znajduje (ang. breadcrumbs), gdzie może pójść dalej, skąd przyszedł oraz co system robi w danym momencie. Jako przykład wymienić można stosowane w aplikacjach różnego rodzaju osie czasu, na których jest zaznaczony bieżący etap procesu. Kolejną ważną rzeczą o której wspomina Nielsen jest to, że odbiorca oprogramowania powinien mieć zagwarantowaną kontrolę oraz jak największą swobodę nad wszystkimi wykonywanymi akcjami. Kiedy zabłądzi np. na stronie internetowej, warto dać mu możliwość powrotu do strony głównej z dowolnego miejsca. Poczuje się pewniej i bezpieczniej, jeśli będzie mógł wracać do poprzednich ekranów oraz kiedy będzie mógł cofnąć swoje niewłaściwe wybory, np. zechce usunąć omyłkowo dodany załącznik czy towar z koszyka albo zatrzymać proces kompilacji kodu, który mógłby trwać zbyt długo. Dobrze zaprojektowana aplikacja pozwoli użytkownikowi naprawić swoje błędy. Im więcej zmian da się cofnąć, tym lepiej. Warto zwrócić na to uwagę podczas testów. Kolejnym dobrym przykładem ochrony przed pomyłkami może być stosowanie standardowych konwencji projektowych. Jeśli system zapewnia projekt niezgodny z innymi podobnymi systemami, użytkownik może się zdezorientować i nie może zrozumieć, jakie działania podjąć, tym samym naraża się na popełnienie błędu. Co by było, gdyby na przykład komunikat “Twój

wniosek został rozpatrzony negatywnie" został wyświetlony w kolorze zielonym zamiast np. w czerwonym? Odbiorca mógłby zwątpić czy faktycznie jego wniosek nie przeszedł, czy pojawił się defekt. Za inny przykład niewłaściwego użycia konwencji niech posłuży witryna mobilna Southwest korzystająca z modelu wyszarzania dat z przeszłości, co oznacza, że nie można ich wybrać podczas rezerwacji lotu. Do pewnego momentu wykorzystywała również ten sam schemat dla dat następnego miesiąca, co sugerowało niedostępność.

Estetyka interfejsu użytkownika (user interface aesthetics)

Wygląd GUI pozwala na zwiększenie (lub zmniejszenie) odczuwania przez odbiorcę oprogramowania satysfakcji z jego użytkowania. Jest to dość subiektywna kwestia. Na estetykę interfejsu ma wpływ po pierwsze dobór kolorystyki poszczególnych elementów, która powinna być dopasowywana do charakteru programu. Należy unikać stosowania „gryzących się" barw. Do zbadania koloru elementu na stronie możemy użyć różnego rodzaju rozszerzeń przeglądarki (np. Color Picker w Chromie). Dzięki temu możemy np. zweryfikować czy barwy użyte w poszczególnych komponentach aplikacji są poprawne. Kolejnym elementem GUI są kroje oraz rozmiary czcionek. Należałoby sprawdzić czy są one czytelne, odpowiednie do specyfiki aplikacji, czy nie są zbyt wyszukane, zbyt małe, zbyt duże. W narzędziach deweloperskich przeglądarki, w kodzie CSS, możemy zweryfikować wielkości oraz rodzaje użytych w czcionek. Warto się także przyjrzeć formie graficznej komponentów GUI:

- układowi elementów na ekranie (czy jest czytelny, przejrzysty, a poszczególne elementy interfejsu są łatwo rozpoznawalne),
- logice ich umiejscowienia,
- zagęszczeniu (czy ekran nie jest przeładowany nadmiarowymi elementami),
- animacjom podczas wykonywania przez użytkownika akcji,
- odpowiednim odstępom (zwłaszcza dla urządzeń mobilnych).

Wyobraźmy sobie, że w aplikacji bankowej wybór rachunku obciążanego odbywa się za pomocą obiektu radio button. Zasadniczo bardziej atrakcyjnym rozwiązaniem jest użycie listy rozwijanej, gdyż w ten sposób zmniejszymy liczbę elementów strony oraz z pewnością przyspieszy to obsługę formularza. Tester powinien patrzeć krytycznym okiem na funkcjonalność, zastanawiając się czy jest zaimplementowana w optymalny, z punktu widzenia UX, sposób. Warto porównywać projekt graficznego interfejsu użytkownika z najlepszymi wzorcami, na przykład regułami Material Design, które wyznaczają nowe trendy. Ten opracowany przez Google w 2014 roku język projektowania wykorzystuje układy oparte na siatce, responsywne animacje i przejścia, wypełnienia oraz efekty głębi, takie jak oświetlenie i cienie. Skoro jesteśmy przy inspiracjach, warto wspomnieć o serwisach takich jak awwwards.com czy thebestdesigns.com, gdzie znajdziemy wiele wzorców, z którymi możemy porównywać testowaną przez nas witrynę. Na koniec tej sekcji należałoby wymienić jeszcze testy wizualne (ang. visual testing), które automatyzują proces wykrywania i przeglądania zmian wizualnego interfejsu użytkownika. Np. wykorzystując aplikację percy.io czy specjalne pluginy do frameworka Cypress, uzyskamy wgląd w zmiany wizualne przy każdej zmianie kodu albo upewnimy się, że aplikacja jest prawidłowo wyświetlana.

Dostępność (accessibility)

Cecha ta określa stopień, w jakim oprogramowanie może być wykorzystywane przez ludzi o różnych cechach psychofizycznych, zwłaszcza z różnymi poziomami niepełnosprawności. Testowanie

dostępności polega na weryfikacji, czy produkt jest przystosowany do odpowiedniej grupy użytkowników. Na przykład, jeśli program ma być dostosowany do osób niedowidzących, powinien umożliwiać wykorzystanie kontrastowych kolorów oraz powiększenie rozmiaru czcionki. W niektórych aplikacjach do bankowości internetowej czy na stronach rządowych obsługujących profil zaufany można spotkać się z funkcją wysokiego kontrastu. Z kolei dostosowanie do potrzeb osób niewidomych może np. wymagać, aby wyświetlane komunikaty mogły być odczytane przez specjalistyczne oprogramowanie przetwarzające tekst pisany na głos lub mogły na bieżąco generować wersję napisaną w piśmie Braille'a (tzw. monitory brajlowskie). W przypadku osób cierpiących na dysfunkcje motoryczne odczytywanie informacji ze stron www może być wspomagane przy pomocy różnego rodzaju manipulatorów. Prostym ułatwieniem wydaje się także zastosowanie skrótów klawiaturowych. Ciekawą opcją znajdującą się w narzędziach deweloperskich przeglądarki (np. w Chromie) jest możliwość wygenerowania audytu witryny, dzięki któremu możemy m.in. uzyskać dane dotyczące dostępności. Dowiemy się na przykład, czy kolory tła i pierwszego planu mają wystarczający współczynnik kontrastu lub czy elementy formularza zawierają powiązane etykiety, co zapewnia prawidłową obsługę przez technologie wspomagające, takie jak czytniki ekranu. Poza tym, w raporcie z audytu znajdziemy informacje dotyczące wydajności czy SEO. Jeśli potrzebujemy jeszcze bardziej szczegółowych i wyczerpujących danych na temat dostępności, możemy skorzystać z pluginu axe dla przeglądarki Chrome. Nie trzeba chyba wspominać o tym, jak bardzo nasze oprogramowanie wyróżni się na tle produktów konkurencji, gdy będzie ono dostępne dla szerszej, szczególnej grupy odbiorców, np. osób z niepełnosprawnością.

Rutynowe testy z użytecznością w tle

Codziennością testera jest wykonywanie testów funkcjonalnych, w których zwraca się przede wszystkim uwagę na to, czy dana funkcjonalność lub cała aplikacja działają poprawnie. Testy funkcjonalne są z natury obiektywne, zaś testy użyteczności – mocno subiektywne. Myślę, że z tego powodu zapomina się czasem o tym, iż oprogramowanie ma być też po prostu ładne dla oka, przyjemne w odbiorze, wspierające użytkownika w procesie korzystania czy chroniące przed pomyłkami. W jaki sposób zauważać takie, z pozoru nieistotne, szczegóły? Myślę, że na początku warto uświadomić sobie, jakie elementy systemu składają się na doświadczenia użytkownika. Jeśli sobie to uzmysłowimy, łatwiej będziemy dostrzegać np. niespójne elementy interfejsu, zbyt jaskrawe lub niepasujące kolory. Warto wejść w buty użytkownika aplikacji i zobaczyć czego mu brakuje, co przeszkadza lub irytuje. W prologu do testerskiej Biblii, czyli sylabusa ISTQB, można przeczytać że celem testowania jest m. in. zapewnienie jakości. Zatem nie tylko obsługa defektów, ale właśnie dołożenie wszelkich starań, żeby oprogramowanie wykazywało się wysoką jakością. W dużej mierze składają się na nią wymagania niefunkcjonalne. Jednak można sobie pomyśleć: „Przecież mam testować główne funkcjonalności dla danego systemu, a nie przejmować się jakimiś czeskimi błędami czy sierotkami." Jednakże wiszące wyrazy albo inne drobiazgi wpływają na obraz aplikacji i tym samym na jakość. Pamiętam, że podczas mojej pracy robiłem masowe zrzuty ekranu sierotek i wysyłałem je programistom, którzy je regularnie poprawiali. Niby mała rzecz, a cieszyła, efekt był widoczny. Warto pamiętać o tym problemie już na etapie implementacji kodu, kiedy to twarda spacja przesunie nam samotny wyraz do nowej linii. Uważam że cechą dobrego testera jest, poza sprawdzaniem najważniejszych funkcji oprogramowania, również dostrzeganie z pozoru nieistotnych, mało widocznych szczegółów związanych z użytecznością. Np. zauważy on, że dodając

ogłoszenie sprzedaży na portalu ogłoszeniowym aplikacja nie podpowiada nam kategorii na podstawie wpisanego tytułu, a mogłaby to zrobić, bo takie rozwiązania widać w produktach konkurencji. Tylko po co ta cała użyteczność? Gdyby ułożyć bardzo uproszczoną zależność opisującą cykl tworzenia oprogramowania, to wyglądałaby tak: designer zaprojektuje, programista je zakoduje, natomiast tester – przetestuje. Skoro jest on na końcu tego łańcucha, zweryfikuje efekty pracy nie tylko dewelopera ale i designera.

Użyteczność ma znaczenie

Podczas codziennych testów oprogramowania warto pamiętać, że posiada ono postrzeganą subiektywnie cechę, której interpretacja dokonywana jest przez każdego użytkownika. Może to powodować, że będzie ona niewidoczna na pierwszy rzut oka, a przez to nieraz pomijana. Mając świadomość poszczególnych komponentów użyteczności takich jak: dostęp, ochrona przed błędami użytkownika, estetyka GUI, łatwość użycia oraz nauki, z pewnością łatwiej przyjdzie nam wyłapywać niedoskonałości z punktu widzenia doświadczenia użytkownika. Bez wątpienia wpłynie to na końcową jakość produktu, bo jak wiadomo, nie samymi błędami funkcjonalnymi tester żyje. Ostatecznie chodzi przecież o to, żeby użytkownik był zadowolony, w myśl powiedzenia „Klient nasz pan”. **Na koniec dodajmy, że testy użyteczności (choć w nieco innej formie niż te dla aplikacji) mają ogromne znaczenie w świecie Internetu Rzeczy, ze względu na s**komplikowaną architekturę i wyjątkowy charakter systemów IoT. **Szacuje się, że w 2020 roku liczba urządzeń podłączonych do sieci przekroczy 25 miliardów. I niewątpliwie, ktoś będzie musiał je przetestować.**

Literatura:

- Adam Roman, *Testowanie i jakość oprogramowania*
- Rafał Pawlak, *Testowanie Oprogramowania. Podręcznik Dla Początkujących*
- *ISTQB Certyfikowany Tester - Poziom Podstawowy 2018 – Sylabus*
- *TestArmy QA Trendbook — 2019 Trends*
- dev.to/izzet/error-prevention-one-of-the-usability-heuristics-2l4b