



ABC TESTOWANIA OPROGRAMOWANIA

Waldemar Szafraniec
Dorota Gabor

©2020 Waldemar Szafraniec
Kraków
www.wyszkolewas.com.pl

Książka jest materiałem bezpłatnym.

Spis treści

Od Autorów.....	1
Kim jestem?.....	2
Wyszkołęwas w liczbach.....	3
Dla kogo jest ten e-book?.....	4
Po co w ogóle testować?.....	5
Jaka jest rola testowania?.....	6
Testowanie a Jakość.....	7
Jak się ma testowanie do Jakości?.....	7
Jak dużo lub długo trzeba testować?.....	8
Ogólne zasady testowania.....	9
Testowanie ujawnia usterki, ale nie może dowieść ich braku.....	9
Testowanie gruntowne jest niewykonalne.....	9
Wczesne testowanie.....	10
Kumulowanie się błędów.....	10
Paradoks pestycydów – Moja nazwa Paradoks świątecznych porządków	11
Testowanie zależne jest od kontekstu.....	12
Przekonanie o braku błędów jest błędem.....	12
Proces Testowy.....	14
Fazy procesu testowego.....	14
Cechy dobrego testera.....	17
Modele wytwarzania oprogramowania.....	20
Model Kaskadowy (Waterfall).....	20
Metodyki zwinne Scrum.....	21
Typy testów.....	25
Testy funkcjonalne.....	25
Testy нефunkcjonalne.....	25
Testy strukturalne.....	26
Testy regresywne.....	26
Poziomy testów.....	27
Testy jednostkowe.....	27
Testy integracyjne.....	28
Testy systemowe.....	28
Testy akceptacyjne.....	28
Czym jest defekt / usterka?.....	29
Rodzaje błędów.....	29
Elementy zgłoszenia defektu.....	31
Zgłaszanie błędu na przykładzie programu Mantis.....	35
Cykl życia defektu.....	36
Standardowe atrybuty scenariusza testowego.....	39

<u>Przypadek testowy.....</u>	<u>41</u>
<u>W jakim celu piszemy przypadki testowe?.....</u>	<u>41</u>
<u>Przypadek niskopoziomowy.....</u>	<u>42</u>
<u>Przypadek wysokopoziomowy.....</u>	<u>43</u>
<u>Estymacja czasu pracy.....</u>	<u>44</u>
<u>Dane potrzebne by dobrze wycenić zadanie.....</u>	<u>44</u>
<u>Techniki estymacji czasu pracy.....</u>	<u>48</u>
<u>Usprawnienia w pracy testera.....</u>	<u>52</u>
<u>Code review.....</u>	<u>52</u>
<u>Testy automatyczne.....</u>	<u>52</u>
<u>Ustalenie norm jakości.....</u>	<u>53</u>
<u>Teorię już znasz, co dalej.....</u>	<u>54</u>
<u>Lista najczęściej padających pytań na rozmowie na testera</u> <u>oprogramowania.....</u>	<u>54</u>
<u>Ćwicz testowanie.....</u>	<u>57</u>

OD AUTORÓW

Drogi czytelniku, w proces pisania tego ebooka włożyliśmy swoje siły, czas a co za tym idzie pieniądze. Dlatego prosimy, abyś szanował nasz wysiłek i starania i **nie udostępniał tego materiału nikomu bez naszej wiedzy**.

Dziękujemy,
Waldemar Szafraniec,
Dorota Gabor

KIM JESTEM?



Nazywam się Waldemar Szafraniec. Jestem trenerem, moją specjalizacją jest szkolenie przygotowujące do zawodu testera oprogramowania. Prowadzę stronę firmową i bloga pod adresem:

www.wyszkolewas.com.pl

Na co dzień jestem również testerem oprogramowania obecnie pracuję na stanowisku QA Test Lead. Karierę rozpocząłem w 2012 roku, z biegiem lat awansowałem, do moich obowiązków dołączyły szkolenie nowo przyjętych pracowników a także rekrutowanie ich. I tak właśnie z biegiem czasu zdobyłem doświadczenie i wiedzę potrzebne do tego by zacząć w ramach własnej firmy przygotowywać ludzi do podjęcia pracy jako tester oprogramowania.

WYSZKOLEWAS W LICZBACH

Na dzień dzisiejszy.

- Przeszkolonych 47 osób, które chciały przebranżowić się na testera oprogramowania.
- Przeszkolone 4 osoby, które pracowały jako tester w momencie rozpoczęcia szkolenia.
- Przeszkolonych 8 osób w ramach szkolenia dla firm.
- 100% dobrych rekomendacji na Facebook, LinkedIn i Google.
- 30 osób, które przeszkoliłem i nie miały wcześniej pracy jako tester oprogramowania ma teraz pracę w tym zawodzie.
- 38 wpisów na blogu.
- 2300 czytelników miesięcznie na blogu.
- Fanpage polubiony przez ponad 300 osób.

DLA KOGO JEST TEN E-BOOK?

W internecie znaleźć możesz wszystko, problem polega na tym, że osoba zaczynająca nie wie od czego zacząć więc czyta wszystko co znajdzie i zaczyna się bałagan.

Ten materiał to wstęp do teorii testowania. Jest dla Ciebie jeśli właśnie zaczynasz naukę. Jest dla Ciebie jeśli chcesz sprawdzić czy znasz już podstawy. Jest dla Ciebie jeśli właśnie skończyłeś się uczyć teorii i nie wiesz co dalej. Jest dla Ciebie jeśli zastanawiasz się czy zawód testera jest dla Ciebie.

PO CO W OGÓLE TESTOWAĆ?

Testujemy w celu zmierzenia jakości danego produktu.

Dzięki temu pomiarowi wiemy, czy produkt nadaje się do użytkowania / oddania klientowi.

W zależności od rodzaju wytwarzanego oprogramowania różne będą kryteria decyzyjne czy nadaje się on do wdrożenia.

Przykłady:

1. Aplikacja umożliwiająca klientowi projektowanie własnych wizytówek powinna podczas drukowania zawsze odwzorowywać dokładnie to, co klient zaprojektował.
2. Strona główna firmy nie powinna mieć literówek, ponieważ jest formą reprezentacji firmy w oczach klienta tej firmy.

Żyjemy w czasach dużej konkurencyjności na rynku oraz łatwego rozprzestrzeniania się opinii klientów o naszych produktach. Dlatego aplikacje, które zawierają wiele błędów oraz posiadają swój odpowiednik na rynku, nie będą używane przez wielu użytkowników. **Opinia**, jaką ktoś sobie wyrobi o nas, **może rzutować później na potencjalnym zysku firmy**, którą reprezentujemy lub nas samych.

JAKA JEST ROLA TESTOWANIA?

Testy same w sobie nie zwiększają jakości aplikacji. Dzieje się tak dlatego, że zgłoszone defekty muszą być poprawione. W przypadku rygorystycznego testowania zarówno systemu, jak i dokumentacji testowej nasze działania mogą spowodować odnalezienie problemów (defektów), które następnie można poprawić.

Jeśli to się stanie przed wdrożeniem wersji systemu na środowisko produkcyjne, to nasze działania wpłyną na podniesienie się jakości systemu.

Testy aplikacji mogą być wymagane przez kontrakt zawarty między firmą wytwarzającą oprogramowanie a firmą zamawiającą. Mogą też być konieczne ze względu na uwarunkowania prawne lub standardy w danej gałęzi przemysłu. Mogą też być dyktowane tym, że firma produkująca oprogramowanie chce, by było ono jak najlepsze, by wielu ludzi go kupiło.

TESTOWANIE A JAKOŚĆ

To jakość produktu sprawia, że użytkownik chce go używać / posiadać. Jest również wyznacznikiem tego, czy chcemy z danym materiałem pracować, czy nie. Dla producenta jakość produktów wyznacza czy będzie miał klientów i sprzeda produkt.

TESTOWANIE JEST ZBIOREM CECH WYROBU,
USŁUGI, PROCESU, KTÓRY ZADOWOLI
WSZYSTKIE ZAINTERESOWANE STRONY.

Jak się ma testowanie do Jakości?

Po wykonaniu testów możemy wyciągnąć wnioski o Jakości produktu. Wnioski te mogą być ogólnym odczuciem co do produktu (czy jest on dobry, czy nie), ale też mogą być dokładnie zmierzone, gdy wygenerujemy raporty z testów. Wyniki testów dostarczają zatem informację o Jakości, w pewien sposób możemy ją tym samym zmierzyć. Czyli wygenerować raporty mówiące o tym ile błędów zostało

zgłoszonych, ile wśród nich jest naprawionych, ile przypadków przeszło testy itd. itp.

Wynik testów pozwala budować zaufanie co do jakości oprogramowania. Brak błędów wysokich i krytycznych mówi o tym, że ważne funkcjonalności działają. Zatem Jakość może być wystarczająca do wdrożenia produktu. Z drugiej strony duża liczba błędów wysokich świadczy o niskiej Jakości oprogramowania.

Każdorazowo po przeprowadzeniu testów powinniśmy je podsumować. Powinno się zastanowić nad rzeczami, które wyszły i tymi, które nie udało się zrobić dobrze, a jednak można je ulepszać w przyszłości.

Jak dużo lub długo trzeba testować?

Powinniśmy testować tak długo, aż będziemy w stanie określić w sposób jednoznaczny jaki jest stan aplikacji.

Takie jest założenie idealne, w świecie realnym mamy budżet i klienta, który czeka na produkt. Dlatego zwykle przyjmuje się dla każdego projektu kryteria zakończenia testowania.

Testowanie powinno dać wszystkim zainteresowanym informacje wystarczające do podjęcia decyzji czy produkt nadaje się do dalszego rozwoju lub wdrożenia go na produkcję.

Przykładowe kryteria zakończenia testów:

- 100% test case zostało wykonanych i przeszło pozytywnie testy.
- 95% test case przeszło pozytywnie.
- Wszystkie błędy powyżej priorytetu wysokiego zostały poprawione.
- Raport z testów został zaakceptowany.

OGÓLNE ZASADY TESTOWANIA

Testowanie ujawnia usterki, ale nie może dowieść ich braku

Nie ma oprogramowania, które nie zawierałoby ani jednego błędu, podczas wykonywania testów rzeczą naturalną jest znajdowanie tych usterek.

Jednocześnie należy pamiętać, że nie znalezienie żadnego błędu w aplikacji nie jest jednoznaczne z tym, że aplikacja jest bezbłędna.

Testowanie gruntowne jest niewykonalne

Sama zasada zakłada, że nie da się przetestować aplikacji w 100%, wyjątkiem są aplikacje bardzo małe.

Moim zdaniem możliwe jest pokrycie aplikacji w 100% testami w przypadku, gdy mamy odpowiedni budżet i wystarczającą ilość czasu. Tylko, trzeba przy tym wiedzieć, że taki budżet będzie bardzo duży i czas wytwarzania produktu będzie bardzo długi. Dzieje się tak dlatego, że każdorazowo poprawiając zgłoszone błędy, programiści ingerują w kod, a to może powodować znalezienie nowych błędów. Równocześnie im więcej iteracji testowych będzie, tym priorytety błędów powinny być niższe. Z czasem powinniśmy zbliżyć się do momentu, gdy uznamy, że cała aplikacja powinna działać.

Wczesne testowanie

Wczesne testowanie jest bardzo ważne! Im wcześniej znajdziemy problem w aplikacji a najlepiej w dokumentacji, tym mniejszy koszt trzeba będzie przeznaczyć na wyeliminowanie go.

PRZYKŁADOWO, ZAŁOŻMY, ŻE PLANOWANE JEST UTWORZENIE APLIKACJI DLA KANTORA. W DOKUMENTACJI W ROZDZIALE DOTYCZĄCYM ZASAD PRZEWALUTOWANIA CZYTAMY, ŻE KWOTY MAJĄ BYĆ LICZONE PODCZAS PRZEMNAŻANIA Z PRECYZJĄ CZTERECH MIEJSC PO PRZECINKU. WYNIK MNOŻENIA MA BYĆ ZAOKRĄGLONY W DÓŁ DO DWÓCH MIEJSC PO PRZECINKU. W DOKUMENTACJI JEST TEŻ WZÓR A W NIM ZAPIS, Z KTÓREGO WYNIKA, ŻE NAJPIERW JEST ZAOKRĄGLENIE WALUTY DO DWÓCH MIEJSC PO PRZECINKU, A NASTĘPNIE PRZEMNOŻENIE.

Jeśli na tym etapie, gdy aplikacja istnieje tylko na papierze pójdziemy do analityka, by wyjaśnić nieścisłość, to poprawienie tego błędu będzie kosztować tyle co czas poświęcony przez analityka na uzgodnienie, który zapis jest właściwy przy wyliczeniach. W sytuacji, gdy algorytm wyliczania jest już zaimplementowany, koszt naprawy błędu to czas analityka by wyjaśnić jak ma wyglądać liczenie + czas testera na testowanie + czas programisty na weryfikację błędu + (ewentualnie jeśli zaimplementowany w kodzie wzór jest tym niewłaściwym) czas na poprawienie kodu + czas na retesty.

Kumulowanie się błędów

Bardzo często zdarza się tak, że w danym module występuje więcej niż jeden błąd. Wówczas, zanim zgłosimy kilka podobnych błędów, powinniśmy się zastanowić, czy poprawienie jednego nie naprawi pozostałych.

PODCZAS PRACY W IT WAŻNY JEST ZYSK, JAKI WYGENERUJEMY POPRZECZ SWOJĄ PRACĘ, NALEŻY WIEC UWAŻAĆ, ABY NIE ZGŁASZAĆ BŁĘDÓW, KTÓRE ZOSTANĄ NAPRAWIONE PRZY OKAZJI ZGŁOSZENIA, KTÓRE WCZEŚNIEJ ZAŁOŻYLIŚMY.

PRZYKŁADOWO, PRZYCISK X JEST WIDOCZNY TYLKO DLA UŻYTKOWNIKÓW Z UPRAWNIENIEM Y.

PO NACIŚNIĘCIU NA PRZYCISK POWINIEN POJAWIĆ SIĘ FORMULARZ.

UŻYTKOWNIK BEZ UPRAWNIENIA Y KLIKA W PRZYCISK CO JUŻ JEST BŁĘDEM, A NASTĘPNIE WIDZI FORMULARZ, KTÓRY JEST ŹLE SFORMATOWANY (CO TEŻ JEST RÓWNIEŻ BŁĘDEM).

SPRAWDZAMY! UŻYTKOWNIK Z ROLĄ Y KLIKA W PRZYCISK X, A NASTĘPNIE WIDZI FORMULARZ, KTÓRY WYGLĄDA DOBRZE.

Wniosek: Znaleźliśmy dwa błędy, z czego naprawa jednego nie ma sensu, ponieważ użytkownik bez uprawnień nie powinien mieć dostępu do tej funkcjonalności. Dlatego też w tym przypadku zgłaszamy błąd dotyczący możliwości kliknięcia w Przycisk X bez uprawnień Y.

Paradoks pestycydów – Moja nazwa Paradoks świątecznych porządków

Zasada mówiąca o tym, że jeżeli będziemy powtarzali tego samego typu testy, to w pewnym momencie dojdzie do sytuacji, w której usterki „uodpornią” się na nasze działania.

założmy, że mamy sytuację, w której przy każdej iteracji testujemy aplikację na podstawie tych samych przypadków testowych. Zatem przy każdej iteracji wykonujemy te same testy. Jeśli przypadki testowe

pokrywają 65% aplikacji, to pozostałe 35% jest „odporna” na nasze testy, ponieważ jej nie testujemy.

PRZYKŁAD: ZAŁÓŻMY, ŻE NASZĄ APLIKACJĄ JEST NASZE MIESZKANIE. W KAŻDĄ SOBOTĘ SPRZĄTAMY MIESZKANIE TZN. ODKURZAMY, ŚCIERAMY KURZE, MYJEMY PODŁOGI. PO SKOŃCZONEJ PRACY MIESZKANIE WYGLĄDA NA CZYSTE, JESTEŚMY ZADOWOLENI. STAN NASZEGO COTYGODNIOWEGO ZADOWOLENIA TRWA STAŁE DO CZASU, AŻ ZBLIŻYMY SIĘ DO ŚWIĄT, URODZIN, IMIENIN, ODWIEDZIN TEŚCIOWEJ... WÓWCZAS POMIMO ŻE SPRZĄTALIŚMY TYDZIEŃ W TYDZIEŃ, NAGLE OKAZUJE SIĘ, ŻE NASZE MIESZKANIE JEDNAK JEST BRUDNE, MA KURZ POD ŁÓŻKIEM, OKNA SĄ NIE UMYTE, SZAFKA NIEPOSPRZĄTANA, REGAŁY NIEPOUKŁADANE.

TAK SAMO WYGLĄDA NASZA APLIKACJA, BARDZO CZĘSTO WYKONUJEMY TESTY CZĘŚCIOWE I TYLKO RAZ NA JAKIŚ CZAS TESTUJEMY GRUNTOWNIE WSZYSTKO PRZEZ CO MOGĄ UMYKAĆ NAM BŁĘDY.

Rozwiązaniem problemu pestycydów może być częste aktualizowanie / zmienianie przypadków testowych tak, aby obszary testów się zmieniały, pisanie wysokopoziomowych przypadków testowych lub wykonywanie co jakiś czas testów eksploracyjnych na aplikacji.

Testowanie zależne jest od kontekstu

Każda aplikacja jest inna oraz potrzebuje innego rodzaju testów, aby potwierdzić jej wysoką jakość. Rodzaje wykonanych testów aplikacji powinny być dostosowane do kontekstu, w jakim aplikacja działa. Przykładowo dla aplikacji, która będzie posiadała jednego użytkownika, nie ma sensu robić testów wydajnościowych. Takie testy będą stratą pieniędzy. Dla aplikacji, która działa offline nie ma sensu robić testów z włączonym dostępem do internetu.

Przekonanie o braku błędów jest błędem

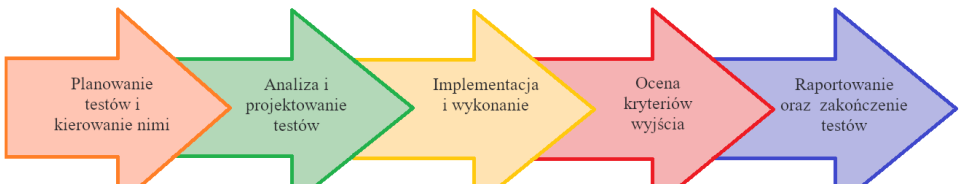
Zgodnie z tą zasadą, nawet jeśli jesteśmy przekonani o braku błędów w

aplikacji, one ciągle mogą tam być. Wynika to z tego, że jesteśmy ograniczeni czasem i budżetem, dlatego nie możemy stale przy każdej iteracji sprawdzić całej aplikacji. Wyeliminowanie błędów jest możliwe, jeśli wrócimy do paragrafu o testowaniu gruntownym i o tym, że testowanie zależne jest od kontekstu. Wówczas jeśli klient ma tylko fundusze, ale również odpowiada mu termin, jaki wyznaczymy na testy oraz ograniczymy się do testów, które mają sens dla danej aplikacji, to moim zdaniem możemy wykluczyć prawie wszystkie błędy z systemu. W celu upewnienia się co do jakości produktu można zlecić audyt firmie zewnętrznej, która potwierdzi jakość naszego oprogramowania.

Zasada ta dotyczy również tego, że nawet jeśli będziemy pewni, że aplikacja działa zgodnie z wymaganiami opisanymi w dokumentacji, to nadal może ona nie spełnić wymagań użytkownika docelowego. Nie będzie wówczas używana.

PROCES TESTOWY

Na proces testowy składają się wszystkie czynności dotyczące pracy testerów, jakie wykonywane są w trakcie wytwarzania oprogramowania. Tak samo, jak wytwarzanie oprogramowania składa się z faz, tak i proces testowy jest na nie podzielony.



Fazy procesu testowego

- Planowanie testów i kierowanie nimi
- Analiza i projektowanie testów
- Implementacja i wykonanie
- Ocena kryteriów wyjścia
- Raportowanie oraz czynności związane z zakończeniem testów

Planowanie testów i kierowanie nimi

Pierwsza faza procesu testowego. Jak sama nazwa podpowiada, w tej fazie następuje planowanie prac, wyznaczenie celu (końca projektu), analiza zasobów potrzebnych, wyznaczenie warunków zakończenia projektu. Określane są metody testów, typy testów jakie zostaną wykonane. W fazie planowania tworzony jest harmonogram prac, który staje się również wyznacznikiem postępu prac w następnych fazach.

Analizowanie i projektowanie testów

Przekształcanie ogólnych, celów testowania w konkretne warunki testowe i przypadki testowe.

Co robimy w ramach tego zadania?

- Przeglądanie podstaw testów (specyfikacja wymagań, architektury, interface).
- Ustalenie poziomu pokrycia testów.
- Identyfikacja i priorytetyzacja warunków testowych.
- Opracowanie przypadków testowych.
- Projektowanie środowiska testowego.

Implementacja i wykonanie testów

Co robimy w ramach tego zadania?

- Tworzenie przypadków testowych oraz określanie ich priorytetów.
- Przygotowywanie danych testowych.
- Pisanie procedur testowych.
- Tworzenie automatycznych skryptów testowych.
- Tworzenie scenariuszy testowych na podstawie przypadków testowych.
- Wykonywanie przypadków testowych ręcznie lub przy pomocy narzędzi automatycznych.
- Logowanie wyników wykonania testów (wersja oprogramowania,

narzędzie testowe).

- Porównanie wyników z oczekiwaniami.
- Raportowanie usterek.
- Wykonywanie ponownie czynności testowych: retesty, testy regresywne.

Ocena kryteriów wyjścia

W tej fazie następuje analiza wykonania celów przyjętych w fazie planowania. Warunki zakończenia testów przyjęte w fazie planowania zostają zweryfikowane tak by upewnić się, że wszystko to co zostało przyjęte za kryterium zakończenia jest spełnione. Zakończeniem tej fazy jest utworzenie raportu z testów.

Raportowanie oraz czynności związane z zakończeniem testów

Ostatnia faza procesu testowego, w ramach której następuje podsumowanie tego co miało miejsce w ramach całego procesu testowego.

CECHY DOBREGO TESTERA

Komunikatywność!

Tester musi mieć wysokie zdolności komunikacyjne, ponieważ jako jedna z nielicznych osób w zespole musi komunikować się ze wszystkimi zainteresowanymi w projekcie: analitykiem, kierownikiem zespołu, programistami i bardzo często z klientem. Dodatkowo zgłaszanie błędów często może zostać odebrane jako wytknięcie pomyłki. Dlatego też tester powinien zawsze zgłaszać błędy z pełnym profesjonalizmem.

Skrupulatność!

Tester powinien być skrupulatny, ponieważ musi zwracać uwagę na najdrobniejsze szczegóły nawet w najnudniejszym oprogramowaniu, jakie kiedykolwiek widział. Praca testera jest BARDZO odpowiedzialna!

Umiejętność Analitycznego myślenia!

Tester musi wykazywać się wysokimi zdolnościami analitycznego myślenia. Zadania, które tester dostaje na warsztat często, są bardzo skomplikowane. Jedną z zasad testowania mówi o Wczesnym testowaniu - testowanie zaczynamy już na poziomie otrzymanej dokumentacji, a dzięki umiejętności analitycznego myślenia jesteśmy w stanie nawet na jej poziomie znaleźć błędy!

Pewność siebie i umiejętność podejmowania decyzji.

Jako tester możesz i będziesz poproszony o wyrażenie opinii, o tym czy aplikacja może zostać udostępniona klientowi. Pewność siebie oznacza, że wyrazisz głośno swoją opinię nie zważając na to, że reszta zespołu sugeruje zakończenie i udostępnienie programu klientowi. W takiej właśnie sytuacji trzeba być silnym by powiedzieć „Nie”.

Ważną rzeczą jest również to, że powiedzenie przez testera "Nie" nie oznacza, że tester ma uprawnienia by podjąć decyzję w kontekście wypuszczenia aplikacji. To zadanie Project managera i to jego odpowiedzialność.

Jeżeli tester jest przeciwny i dobrze uzasadnił swoją ocenę jakości oprogramowania, to reszta należy do osób decyzyjnych.

Kreatywność

Cecha ta pozwala na to by podczas testów przewidzieć różne zachowania użytkownika w aplikacji. Dzięki niej tester znajduje błędy tam, gdzie większość ludzi ich nie dostrzega lub po prostu się na nie godzi.

Umiejętność planowania i samodzielność

Wiele firm stawia obecnie na metodykę Scrum, który wymusza pracę zespołową. Wszyscy członkowie zespołu mają ten sam cel jakim jest dostarczenie działającej aplikacji na czas. Mimo, że cel jest wspólny, praca jest samodzielna. Tester musi umieć zaplanować czas pracy tak by zrealizować wszystkie czekające go taski. Planowanie testów i przygotowywanie sobie środowisk do tych testów, pisanie dokumentacji testowych, testowanie i uczestnictwo w spotkaniach to niektóre z tych zadań.

Odporność na stres

Na szczęście nie jest tak, że z nerwów ręce drżą mi każdego dnia. Gdyby tak było, nikt by nie chciał pracować w tym zawodzie. Przykładowo, przyjmijmy, że Sprint trwa dwa tygodnie, zaczyna się planowaniem testów, a kończy wystawieniem wersji na produkcję. Wówczas zwykle pod koniec Sprintu mamy więcej pracy i co za tym idzie więcej stresu. Poziom stresu zależy od atmosfery w zespole, ale i od tego, jak sami zorganizowaliśmy (zaplanowaliśmy) sobie pracę na cały okres Sprintu.

Nastawienie na rozwój

Jest to ważna cecha charakteru testera oprogramowania. Nastawienie na rozwoju widać już w okresie przygotowywania do podjęcia zawodu. Musisz w tym czasie opanować podstawową teorię testowania. Idąc na rozmowy z potencjalnym pracodawcą powinieneś znać odpowiedzi na najczęściej pojawiające się pytania na rozmowach rekrutacyjnych. I na tym nauka się nie kończy, raczej zaczyna. Od pierwszego dnia pracy oczekuje się od ciebie chęci do nauki – narzędzi i testowania. Samorozwój w tym zawodzie jest bardzo ważny, bez niego nie ma awansów, podwyżek i premii.

MODELE WYTWARZANIA OPROGRAMOWANIA

Model Kaskadowy (Waterfall)

W tym modelu wytwarzania oprogramowania kolejne etapy wytwarzania produktu następują jeden po drugim. Nigdy dwa na raz. Proces rozpoczyna się od Planowania, następnie przechodzi się do Analizy, tworzony jest zarys Projektu. W kolejnej fazie przechodzimy do Implementacji, po której następuje dopiero faza Testów. Na sam koniec dochodzimy do etapu Pielęgnacji (utrzymania aplikacji).

Charakterystyczna jest tutaj praca w zespołach testerskich liczących kilka osób razem z kierownikiem testów. W tym modelu wytwarzania oprogramowania częściej występują sytuacje związane z „potyczką z programistą”. Wynika to z podziału na zespoły programistów i testerów, to, że jest silniejsze poczucie zgrania ze swoim zespołem i dążenia w nim do najlepszych wyników przeciw innym zespołom. Należy pamiętać, by wszelkie konflikty starać się rozwiązać poprzez rozmowę i dowody najpierw z samym programistą, a następnie, jeśli nie zadziała przez kierownika zespołu lub starszego stopniem kolegę.

Aplikacja oddana do testów ma wszystkie funkcjonalności. Testerzy testują ją w oparciu o analizę i utworzone przypadki testowe. Po fazie testów następuje faza poprawy błędów przez programistów. Kolejnym

etapem są retesty błędów. Często zdarza się, że w fazie retestów zgłaszane są kolejne „nowe” błędy do aplikacji. Fazy testów i retestów następują po sobie, dopóki stan aplikacji (mierzony ilością błędów wysokich i krytycznych) nie będzie pozwalał na jej udostępnienie. Często publikacja aplikacji jest konieczna ze względu na zbliżające się terminy oddania aplikacji. Wówczas na zasadzie kompromisu wybiera się błędy niezbędne do poprawienia by aplikacja mogła zostać wydana.

Metodyki zwinne Scrum

Po pierwsze chcę zaznaczyć, że Scrum w Polsce jest bardzo rozległym pojęciem, prawie wszystkie firmy nie korzystają w 100% z czystego Scruma tylko adaptują sobie jego idee w jak najkorzystniejszy sposób dla siebie.

Najważniejsze informacje na temat Scrum

Scrum podchodzi do wytwarzania oprogramowania w sposób iteracyjny-cykliczny (cykle mogą trwać od 2 do 4 tygodni maksymalnie)

Scrum master nie jest kierownikiem, jest to osoba, która ma za zadanie rozwiązywać problemy, jakie się pojawiają w trakcie implementacji.

Charakterystyczną rzeczą w Scrumie jest brak jasno zdefiniowanych ról typu: analityk, tester, programista. Założenie jest takie, że wszyscy stanowią zespół i że wszyscy mają być w stanie nawzajem się zastąpić. W praktyce oczywiście każdy zajmuje się tym, co potrafi najlepiej.

Scrum oparty jest o pracę zespołową. Nie ma podziału na testerów, programistów i analityków. Jest zespół, w którym każdy teoretycznie powinien być w stanie zastąpić każdego. Teoretycznie, ponieważ każdy członek zespołu zajmuje się tym, w czym jest najlepszy. Mamy zatem nadal testera, programistę i analityka. Poniżej opisuję role, z jakimi spotkasz się pracując w Scrumie.

Product owner

Jest to przedstawiciel strony klienta. Nie jest to osoba zatrudniona przez klienta (odbiorcę programu), ale osoba, która reprezentuje dobro klienta. Osoba decyzyjna, jeżeli chodzi o priorytetyzację zadań i starania, aby produkt był jak najbardziej wartościowy. Product Owner powinien dbać o

to, aby oczekiwania były zawsze jasno zdefiniowane. Założenia Scruma mówią o tym, że Product owner jest jedyną osobą decyzyjną dla zespołu developerskiego. Jednocześnie osoba zajmująca to stanowisko nie działa autorytarnie, ale również komunikuje się z zespołem na różnego rodzaju spotkaniach (głównie Backlog refinement, ale również może uczestniczyć w Daily scrum, Review czy też Retrospekcji).

Development team

Zespół składający się z programistów, testera/ów, analityka oraz Scrum mastera. Zgodnie z definicją Scruma nie mamy podziału na rolę my jako zespół powinniśmy mieć jak najszersze kompetencje tak, aby móc dostarczyć, jak najlepszy produkt finalny. Kolejną cechą zespołu developerskiego jest samoorganizacja, która oczywiście rośnie wraz z długością współpracy w ramach naszego zespołu. Ostatnią cechą, jaką należy wymienić w kontekście Development teamu jest wspólna odpowiedzialność za tworzony produkt.

Scrum master

Scrum master jest bardzo ciekawą rolą, ponieważ w przeciwieństwie do np. waterfalla nie jest to nasz przełożony, tylko bardziej kolega w zespole. Osoba będąca Scrum masterem powinna dbać o zrozumienie teorii Scruma oraz przestrzeganie przyjętych zasad. Co więcej, jeżeli zespół napotka jakiegokolwiek przeszkody w dowiezieniu zadań to Scrum master jest osobą, która powinna się tymi przeszkodami zająć. Scrum master również powinien zająć się szerzeniem idei Agile w ramach organizacji, w której pracuje.

Spotkania w Scrum

W pracy w Scrum ważną część odgrywa komunikacja. Integralną częścią Scruma są zatem spotkania, które w ramach każdego sprintu powinny mieć miejsce. Poniżej opisane są te, które są istotne w kontekście naszej (testerów) pracy.

Każdego dnia Scruma odbywa się **Daily Scrum** (Standup). Jest to spotkanie, które z założenia powinno trwać maksymalnie 15 minut.

Podczas tego spotkania powinniśmy powiedzieć, co zrobiliśmy wczoraj. Co mamy w planach zrobić dzisiaj oraz czy nic nam nie przeszkadza w wykonaniu naszej pracy. Tester na takim spotkaniu powinien uważnie słuchać, jak idą prace zaplanowane w kontekście trwającego sprintu, aby móc odpowiednio przygotowanym podjąć zadania zbliżające się do etapu, na którym możemy je zacząć testować.

PRZYKŁADOWO:

DZISIAJ PLANUJĘ ROZPOCZĘCIE PRACY NAD MODUŁEM PRZELICZANIA WALUT. W CZORAJSZEGO ZADANIA DOTYCZĄCEGO TESTÓW MODUŁU LOGOWANIA NIE UDAŁO MI SIĘ WYKONAĆ, PONIEWAŻ W APLIKACJI JEST BŁĄD PODCZAS LOGOWANIA.

Kolejnym rodzajem spotkania jest **Backlog refinement** (grooming). Podczas spotkania przeglądany jest Backlog w celu sprawdzenia, czy na pewno wszystkie funkcjonalności (duże zadania nazywane w Scrumie „historyjkami”) są poprawnie zdefiniowane. Odpowiadamy również na pytanie, czy nie należy niektórych z nich usunąć z Backlog’a. Rozmawiamy o tym, czy nie ma nowych potrzeb (funkcjonalności), a co za tym idzie, nie trzeba stworzyć nowych „historyjek”. Jeśli zauważamy, że w ramach wcześniej zdefiniowanych zadań są takie, które są za duże, rozbijamy je na mniejsze. Spotkanie to jest dedykowane Product Ownerowi, Analitykowi oraz Scrum masterowi (często w tym składzie się ono odbywa).

Podsumowaniem Sprintu jest spotkanie nazywane **Retrospekcją**. Podczas tego spotkania staramy się spojrzeć na cały nasz sprint z perspektywy czasu. Patrząc na poprzedni, sprint staramy się rozpisać pozytywne rzeczy, jak i negatywne sytuacje/zdarzenia, które się wydarzyły. Plusy są o tyle fajne, że wiemy, co powinniśmy starać się kontynuować, natomiast minusy dają nam możliwość poprawy a tym samym udoskonalenia swoich umiejętności.

NIE POWINIENIEŚ BRAĆ DO SIEBIE MINUSÓW,
KTÓRE ZOSTANĄ ZGŁOSZONE DO WYNIKÓW
TWOJEJ PRACY. POTRAKTUJ JE JAKO OKAZJĘ
DO TEGO, ABY SIĘ ROZWINAĆ.

Demo to spotkanie, na którym prezentujemy rezultat naszej pracy wykonanej w trakcie sprintu. Odbiorcami tego spotkania poza naszym zespołem developerskim jest Product owner. Tester bardzo często jest moderatorem tego spotkania, ponieważ tester spośród zespołu developerskiego jest jedną z nielicznych osób, które znają cały zakres prac wykonanych (w końcu je przetestował) w ramach całego Sprintu.

Podczas **Planowania** staramy się całym zespołem zaplanować pracę na przyszły Sprint. W ramach tego spotkania wypełniamy Sprint Backlog zadaniami z Product Backlog'a. Lista powstaje na podstawie kilku czynników: velocity (prędkość) oraz capacity (pojemność), które są mierzone na podstawie Story pointów, czyli miary określającej skomplikowanie, czasochłonność związane z danym zadaniem. Story Pointy nadajemy podczas estymacji czasu pracy w tym wypadku za pomocą planning pokera, o którym przeczytasz w rozdziale o estymacji czasu pracy.

TYPY TESTÓW

Testy funkcjonalne

Testy polegające na weryfikacji wymagań funkcjonalnych aplikacji. Najprostszym językiem tłumacząc, sprawdzają, co robi dany moduł i czy wynik tego, co robi jest poprawny. Podczas wykonywania tych testów wykorzystuje się techniki czarno skrzynkowe, które bazują na specyfikacji, bez wnikania w kod programu. W trakcie testów tester nie ma dostępu i nie czyta kodu źródłowego aplikacji.

Testy нефunkcjonalne

Odpowiedzialne są za sprawdzanie „jak dobrze” działają dane komponenty. Podczas testów нефunkcjonalnych możemy testować niezawodność, efektywność, obciążenie systemu lub bezpieczeństwo aplikacji. Podobnie jak podczas testów funkcjonalnych, do wykonania tych testów stosujemy głównie techniki czarno skrzynkowe. W celu przeprowadzenia testów нефunkcjonalnych wymagane jest najczęściej dedykowane środowisko testowe, czasem nawet specjalistyczny sprzęt czy oprogramowanie.

Testy strukturalne

Zwane są inaczej testami białą skrzynkowymi, co oznacza, że testom poddawany jest kod źródłowy aplikacji (tester zna kod). Ich zadaniem jest przetestowanie możliwie każdej ścieżki wykonania testowanego komponentu. Podczas testów dane podawane są, tak by wymusić na programie przejście możliwie, jak największej ilości ścieżek. W przeciwieństwie do testów funkcjonalnych, testy te nie wykryją czy wszystkie funkcjonalności wymagane w aplikacji są zaimplementowane. Dzieje się tak, ponieważ testom poddany jest istniejący kod. Dzięki tym testom upewniamy się, że wszystkie elementy oprogramowania zostały pokryte przez testy.

Testy regresywne

Testy regresywne najczęściej dotyczą całego systemu i wykonywane są w oparciu o przypadki testowe.

W przypadku testów regresywnych sprawdzeniu poddawany jest cały system. Dlatego testy dla dużych aplikacji trwają nawet kilkanaście osobodni (jeśli nie dla aplikacji nie utworzono testów automatycznych).

Ze względu na koszty związane z testami regresywnymi wykonywane są one wówczas, gdy zmiany zrobione w systemie wymagają weryfikacji całej aplikacji.

Testy regresywne wykonujemy by potwierdzić, że zmiany dokonane w aplikacji nie spowodowały powstania błędów.

POZIOMY TESTÓW

Testy jednostkowe

Polegają na testowaniu pojedynczych modułów (komponentów) oprogramowania. Testom podlegają najmniejsze części oprogramowania (np. pojedyncze funkcje, klasy, moduły). Testy jednostkowe powinny być wykonywane w izolacji od innych modułów. Dlatego stosuje się zaślepki dla komponentów, od których zależy testowany moduł aplikacji. Zaślepki są tak napisane, by zawsze zwracały do aplikacji określone wartości, tak jakby aplikacja działała w połączeniu z modułem zaślepionym. Na tym poziomie testów ważne jest testowanie wartości brzegowych. Testując weryfikujemy jak najwięcej różnych ścieżek przejścia przez dany moduł, najlepiej wszystkie (na podstawie wartości brzegowych).

PRZYKŁADOWO:

W SPECYFIKACJI CZYTAMY, ŻE WIEK REJESTROWANEGO DO SYSTEMU UŻYTKOWNIKA MA BYĆ MIĘDZY 21 A 40 LAT.

WARTOŚCI BRZEGOWE, DLA KTÓRYCH NALEŻY PRZEPROWADZIĆ TESTY: 20, 21, 22, 39, 40, 41.

Testy integracyjne

Celem testów integracyjnych jest sprawdzenie poprawności integracji pomiędzy danymi komponentami (modułami). Na tym poziomie testów nie powinny być zgłaszane błędy do tego, jak działają pojedyncze części oprogramowania, ale do tego, jak one między sobą współdziałają. Dobrym przykładem testów integracyjnych jest sprawdzanie poprawności łączenia się z bazą danych. Na poziomie testów integracyjnych testowane mogą być mniejsze komponenty takie jak funkcje, klasy, a także całe programy, systemy i integracja naszego systemu z nimi. W testach integracyjnych wyróżnić można metody zstępującą i wstępującą. Pierwsza z nich polega na testowaniu w pierwszej kolejności elementów najwyżej w hierarchii, a następnie niższych rzędów, kończąc na najniższym. Metoda wstępująca działa odwrotnie. Zaczynamy od modułów najniższych rzędów, a kończymy na najwyższym poziomie.

Testy systemowe

Przeprowadzane są na zintegrowanym systemie i mają głównie sprawdzać poprawność wykonania scenariuszy testowych z punktu widzenia użytkownika. Testom podlegają założenia i cele projektu. Testy systemowe uznawane są za najtrudniejsze testy, ponieważ bardzo ciężko je zautomatyzować. Wykonywane są najczęściej przez żywego osobnika i trwają długo.

Testy akceptacyjne

Testy, które wykonywane są w celu akceptacji produktu przez klienta, lub inną uprawnioną do tego osobę. W ramach testów weryfikowane są wyłącznie ścieżki (przypadki testowe) wcześniej ustalone i wybrane przez klienta. Testy akceptacyjne nie mają na celu znalezienie błędów, ale potwierdzenie, że aplikacja spełnia oczekiwania i wszystkie założenia przed wdrożeniem.

CZYM JEST DEFEKT / USTERKA?

Jest to zdarzenie w systemie niezgodne z dokumentacją biznesową lub analityczną.

PRZYKŁADOWO:

W DOKUMENTACJI CZYTAMY: $A + B = 2$.

W APLIKACJI WIDZIMY: $A + B = 3$.

Jak widać na powyższym przykładzie, aplikacja nie jest spójna z dokumentacją. A to znaczy, że należy zgłosić defekt.

Rodzaje błędów

Ze względu na logikę błędów i przyczyny występowania możemy podzielić je na:

- funkcjonalne,
- wydajnościowe,
- bezpieczeństwa,
- wizualne.

Przykład błędu **funkcjonalnego** został opisany powyżej. Jest to błąd dotyczący nieprawidłowego działania logiki aplikacji. Zgłaszany jest, gdy aplikacja nie działa zgodnie z oczekiwaniami opisanymi w dokumentacji.

Błędy **wydajnościowe** zgłaszane są w przypadku gdy aplikacja zaczyna

działać w sposób nieprawidłowy w wyniku obciążenia jej dużą ilością użytkowników. Do tego typu defektów dochodzi również, jeśli aplikacja ma małą ilość zasobów pamięci lub gdy aplikacja nie zwalnia zasobów pamięci. W przypadku błędów wydajnościowych aplikacja zazwyczaj przestaje „odpowiadać” na nasze klikanie w nią i konieczny jest jej restart.

Błędy **bezpieczeństwa** dotyczą możliwości wykonania na aplikacji działań przez użytkowników do tego nieuprawnionych. Mogą one dotyczyć zapisania danych przy pomocy formularza wraz z kodem, który wykona się na bazie danych. A także tego, że dla aplikacji webowej możliwa jest zmiana żądania przesyłanego na serwer, tak by to żądanie wykonało się pomimo zmian.

Błędy **wizualne** dotyczą tego, co widzimy na ekranie korzystając z aplikacji. Są to literówki, błędy ortograficzne, niewyrównane przyciski i inne podobne widoczne na ekranie często bez interakcji użytkownika. Błędy te nie wpływają na poprawność działania funkcjonalności aplikacji, ale na jej odbiór wizualny przez użytkownika.

Nasza praca oceniana jest między innymi po tym, jak dobre błędy zgłaszamy. Jeśli chcielibyśmy popatrzeć na statystyki to dobry tester ma mało błędów as designed i mało błędów, w których dodawane były komentarze. Po części w ocenie tej chodzi, jak ważne są te błędy (jaki mają priorytet), ale przede wszystkim o to, czy nasze zgłoszenia nie sprawiają problemu podczas odtwarzania.

DOBRE ZGŁOSZENIE TO TAKIE, KTÓREGO NIE
TRZEBA WYJAŚNIAĆ, KTÓREGO KROKI
TESTOWE OPISUJĄ DOKŁADNIE JAK ZNALEŻĆ
BŁĄD.

We wszystkich tych staraniach chodzi o to, żeby jedyną frustracją, jaka się pojawi z powodu zgłoszenia była złość na fakt, że błąd jest i trzeba go poprawić, a nie na to, że nie wiadomo, o co w błędzie chodzi.

ELEMENTY ZGŁOSZENIA DEFEKTU

Kategoria / Tagowanie

Kategoria dotyczy rodzaju zgłoszenia. Projekt nie tylko ma błędy, ale też np. pytania do analityka (questions), główne zadania tak zwane (story), pod zadania (subtask).

Powtarzalność

W tej formatce określamy na ile prób, błąd udaje się powtórzyć. Dzięki tej informacji programista wie, że powinien kilka razy podjąć próbę odtworzenia defektu, zanim przyjdzie do nas z pytaniami o to, jak to zrobić. Jest to również ważna informacja dla testerów podczas retestów błędów.

Priorytet

Przeważnie mamy do dyspozycji sześć stopni ważności zgłoszenia. Wymieniając od najmniej ważnego będą to: Trywialny, Mniejszy, Normalny, Wysoki, Krytyczny oraz Blokujący. Wartość w polu Priorytet

ustalamy, odpowiadając sobie na następujące pytania.

- Czy istnieje obejście problemu dzięki, któremu użytkownik może używać aplikacji?
- Czy problem jest globalny i dotyczy całej aplikacji, czy też lokalny i dotyczy funkcjonalności w aplikacji?
- Czy problem dotyczy nowej funkcjonalności, którą chcemy dodać do wersji produkcyjnej?
- Czy trudno jest go powtórzyć i jak wielka jest szansa na to, że “normalny” użytkownik na niego trafi?

Temat / tytuł błędu

Krótki opis mówiący co nie działa w aplikacji. Tytuł powinien w sposób jasny i czytelny mówić co nie działa w aplikacji.

Tytuł błędu podobnie jak pozostałe pola nie powinien posiadać literówek ani błędów ortograficznych. Literówki i błędy ortograficzne sprawią, że nie będzie możliwe wyszukanie błędu po słowie kluczowym na liście błędów zgłoszonych do systemu. Jest to szczególnie ważne, gdy system testowany jest duży, ilość błędów zarejestrowanych w bug tracker jest również duża oraz gdy aplikację testuje więcej niż jeden tester.

Opis

W tym polu wpisujemy dane opisowe błędu. Mogą to być informacje szczegółowe na temat tego, jaki komunikat się wyświetlił. Albo dodatkowe informacje początkowe, jakie muszą zaistnieć, np. jeśli błąd wystąpi tylko wtedy gdy użytkownik wyczyści cash przeglądarki.

Kroki testowe

Moim zdaniem najważniejsza część zgłoszenia błędu. W tym polu opisujemy krok po kroku, tak jakbyśmy tłumaczyli komuś, kto nigdy aplikacji na oczy nie widział, co trzeba zrobić, by błąd odtworzyć. Zasady,

których warto się trzymać:

- Każdy krok opisu umieść w nowej linii.
- Numeruj kolejne kroki.
- Stosuj krótkie proste zdania (niezłożone).
- Pisz w formie drugiej osoby.
- Nazwy pól i przycisków umieść tak jak widnieją w aplikacji (nie odmieniaj) dodatkowo warto je umieścić w cudzysłów.

Najczęstszymi błędami testerów podczas opisania kroków testowych jest niedbałość o szczegóły. Mylne założenie, że przecież wiadomo co trzeba zrobić, by powtórzyć błąd. Podanie w krokach testowych linku do aplikacji i zrzutu ekranu nie sprawia, że każdy będzie w stanie odtworzyć kroki testowe.

Testerze zapamiętaj, programista bardzo często nie zna całej aplikacji. Im dokładniejszy opis błędu, tym mniejsze prawdopodobieństwo na stratę czasu przy próbie jego reprodukcji.

Przypisanie

Zazwyczaj jest to rozwijalna lista, na której wybieramy imię i nazwisko programisty, który powinien zająć się danym zgłoszeniem. Przeważnie nad aplikacją pracuje więcej niż jedna osoba, a sama aplikacja jest całkiem spora. Powoduje to konieczność rozłożenia tejże aplikacji na obszary, za które są odpowiedzialne konkretne osoby. Jeśli zgłaszamy błąd, to staramy się go przypisać do osoby, która odpowiada za dany obszar.

Środowisko

Dla aplikacji WEB podajemy w tym polu nazwę przeglądarki, jej wersję oraz wersję aplikacji.

Jeśli defekt wystąpił w aplikacji mobilnej, to podajemy wersję aplikacji

oraz wersję systemu Android/IOS a czasami też model telefonu.

W przypadku aplikacji na komputery podajemy system operacyjny oraz wersję aplikacji, na której znajduje się błąd.

Załączniki

Wszystkie dodatkowe informacje pomagające programiście namierzenie błędu, w szczególności są to: logi skopiowane z przeglądarki, zrzut ekranu (format .png), filmik (.mp4).

Pamiętaj, że załącznik (zrzut, filmik) powinien pokazywać możliwie jak najwięcej by pomóc odnaleźć błąd. Dobrą praktyką jest zaznaczyć na zrzucie ekranu gdzie jest błąd (najprościej użyć w tym celu Paint i ramką czerwoną otoczyć właściwe miejsce na obrazku).

Filmik powinien pokazać dokładnie kroki testowe pokazujące jak otworzyć błąd. Zaczynasz nagrywać w momencie otworzenia strony, na której jest błąd, następnie wykonujesz kolejne kroki testowe aż do wystąpienia błędu. Pliki zapisujesz w formie mp4. Jeśli firma wymaga tego, kompresujesz pliki (przykładowo używając programu HandBrake), tak by nie przysyłać ich jako FULL HD, gdyż tak duże pliki zaśmiecają dyski niepotrzebnie.

Programów do robienia filmików jest wiele. Zazwyczaj firmy zatrudniające mają swoje wykupione na licencji lub zalecane darmowe. Przykładowo dla urządzeń mobilnych możesz używać aplikacji Mobizen, natomiast do nagrywania filmików z ekranu komputera możesz użyć również darmowego ShareX.

Zrzut ekranu robisz tak, by pokazywał, gdzie jest błąd i co jest błędem. Nie przycinaj go tak, by widoczne było samo pole i komunikat. Zrzut powinien pokazać możliwie cały ekran aplikacji. Dobrą praktyką jest podkreślenie czerwonym kolorem miejsca, gdzie jest błąd. Są programy jak Snagit, które mają wbudowany edytor obrazka umożliwiający dodanie strzałek, kresek, podpisów i wielu innych rzeczy. Do tego samego celu równie dobrze można użyć zwykłego Paint i nim zaznaczyć gdzie jest błąd. To jak starannie zgłosisz błąd, świadczy o Tobie i Twojej pracy, więc dobre nawyki, jakimi jest dokładność, staranność i sumienność są mile

widziane. Darmowe narzędzia do robienia zrzutów to te wymienione przy filmikach: Mobizen i ShareX.

Nagrywając filmiki i robiąc zrzuty pilnuj by nie pokazywać na nich tego co nie chcesz by było pokazane. Przykładowo gdy testujesz aplikację na przeglądarce nie robisz zrzutu, na którym widać otwarte zakładki z Facebook.

Zgłaszanie błędu na przykładzie programu Mantis

Założmy, że masz do przetestowania ekran logowania do aplikacji. Po wpisaniu poprawnego loginu i hasła, kliknięciu zaloguj, otrzymałeś komunikat o błędzie jak na poniższym obrazku.



The screenshot shows a login interface. At the top left is the text "Logowanie". At the top right is a home icon and the text "Logowanie". Below this is a pink error message box that reads "Wystąpił błąd podczas próby logowania.". Underneath the error message are two input fields: "Login" and "Hasło". At the bottom right is a blue button labeled "ZALOGUJ".

Dla upewnienia sprawdzasz jeszcze dwa razy – błąd występuje za każdym razem. Przechodzisz zatem do jego opisu.

*Temat	Komunikat o błędzie po kliknięciu zaloguj.
*Opis	Aplikacja wyświetla błąd „Wystąpił błąd podczas próby logowania” po wpisaniu poprawnych danych użytkownika posiadającego login i hasło w aplikacji.
Kroki, by powtórzyć	<ol style="list-style-type: none">1. Uruchom aplikację.2. Podaj „Login” i „Hasło” istniejącego użytkownika.3. Kliknij „Zaloguj”.4. Strona odświeża się.5. Aplikacja wyświetla błąd „Wystąpił błąd podczas próby logowania”.

CYKL ŻYCIA DEFektu

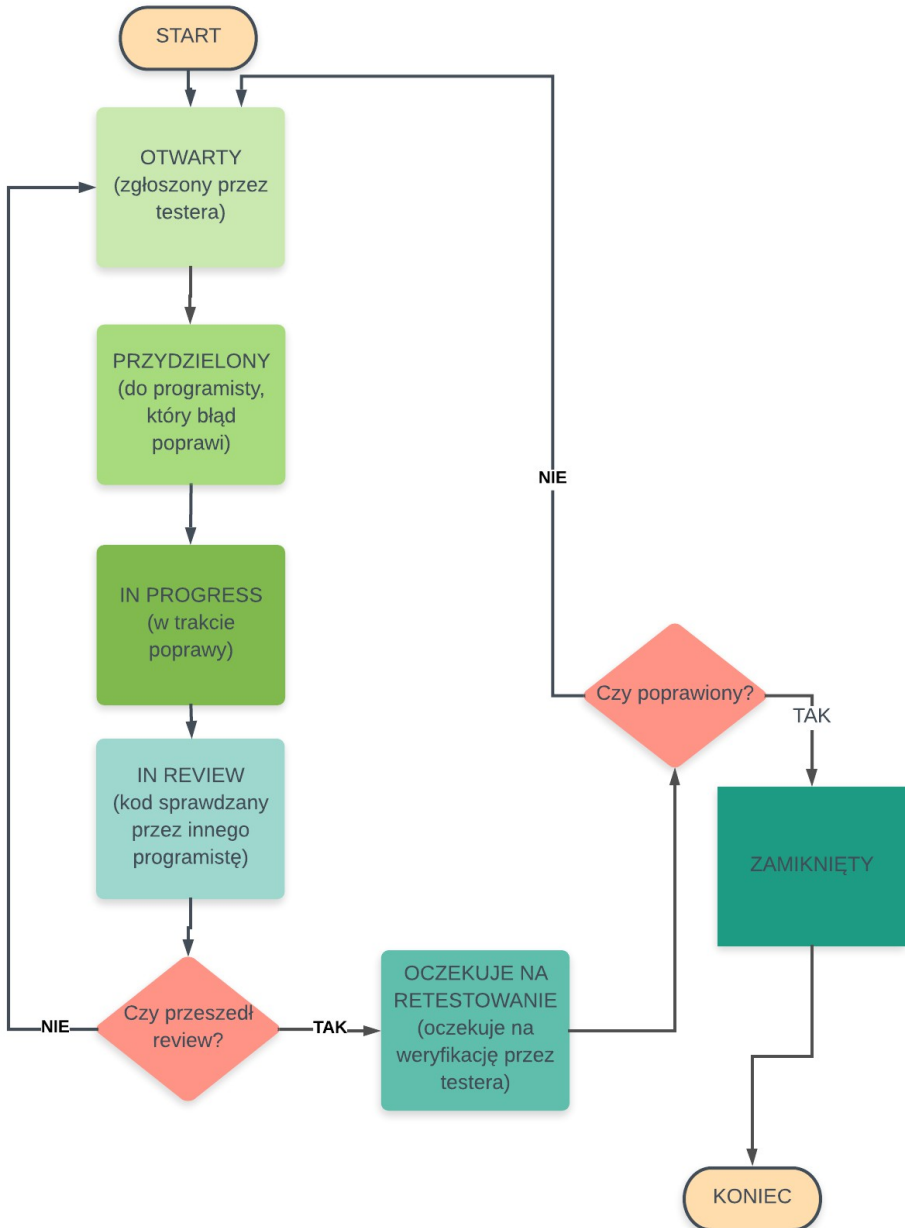
Cykl życia defektu zaczyna się od zapisania go w aplikacji takiej jak Mantis czy inny bug tracker. Tak utworzone zgłoszenie otrzymuje status Otwarty. Znaczy to, że zgłoszenie jest i czeka na liście zgłoszeń. Nikt nic z nim jeszcze nie zrobił.

Jeżeli tester uzupełniając dane o błędzie, wybrał programistę, który powinien błąd poprawić. Błąd automatycznie zmienia status z Otwarty na Przydzielony. Zgłoszenia Otwarte bez przypisanego developera czekają w tym statusie, aż odpowiednia osoba przejrzy je i przypisze do właściwego programisty.

Programista, rozpoczynając pracę nad błędem zmienia jego status na In Progress. Znaczy to dla nas tyle, że błąd jest weryfikowany. Jeśli jest dobrze opisany i jest rzeczywiście błędem, to będzie poprawiony. Jeśli błąd nie jest błędem (zgodnie z dokumentacją program powinien zachować się tak, jak się zachował), developer zamknie zgłoszenie z odpowiednim kodem zamknięcia (zazwyczaj as designed).

Poprawiony przez programistę błąd przechodzi do statusu In Review co oznacza, że inny programista zweryfikuje poprawność kodu i to czy błąd jest poprawiony.

Następnie błąd, który przeszedł review oczekuje na retest testera. Podczas retestu tester wykonując dokładnie kroki opisane w retestowanym zgłoszeniu weryfikuje czy błąd jest poprawiony. Poprawiony błąd zostaje zamknięty. Niepoprawiony wraca do statusu Otwarty.



Retestując wykonujemy dokładnie opisane kroki testowe błędu.

Jeśli przykładowo mamy do weryfikacji błąd o następujących krokach testowych:

1. Uruchom aplikację.
2. Podaj „Login” i „Hasło” istniejącego użytkownika.
3. Kliknij „Zaloguj”.
4. Strona odświeża się.
5. Aplikacja wyświetla błąd „Wystąpił błąd podczas próby logowania”.

I zauważamy, że po podaniu loginu bez hasła aplikacja pozwala na logowanie, to zgłaszamy nowy błąd a ten retestowany zamykamy. Zwykle w sytuacji, gdy mamy wątpliwość, czy błąd wrócić do statusu aktywny, czy nie dobrze jest się zapytać programisty, który ten błąd poprawiał. Zamknięcie błędu oznacza, że został on naprawiony, a zmieniony przez programistę kod przetestowany i zatwierdzony.

STANDARDOWE ATRYBUTY SCENARIUSZA TESTOWEGO

Scenariusz testowy to „dokument określający ciąg akcji umożliwiających wykonanie testu” (źródło sjsi.org).

INNymi SŁOWY SCENARIUSZ TESTOWY
ZBUDOWANY JEST Z PRZYPADKÓW
TESTOWYCH W CELU PRZETESTOWANIA
KONKRETNEJ FUNKCJONALNOŚCI SYSTEMU

Jeśli funkcjonalnością do zweryfikowania będzie logowanie do systemu, wówczas scenariusz testów może zawierać przypadki testowe wymienione poniżej.

- Automatyczne logowanie do aplikacji.
- Logowanie do aplikacji przy użyciu poprawnego loginu i hasła.
- Logowanie nieistniejącego użytkownika.

Tworząc scenariusz testowy, podajemy dla niego tytuł, czyli informację,

jaka funkcjonalność będzie testowana. Następnie Opis, w którym wyjaśniamy, czego dotyczą testy. W kolejnym kroku wymieniamy, co powinno być zrobione w testowanej aplikacji, by możliwe było wykonanie scenariusza. Dla przykładowego scenariusza logowania do aplikacji warunkiem wstępnym może być to, że w systemie istnieje użytkownik i znamy jego login i hasło.

To jakie przypadki będą zawarte w scenariuszu testowym, zależy od tego, jak my testerzy je zaprojektujemy na podstawie wymagań dostarczonych do aplikacji.

Testy scenariuszy testowych niekoniecznie mają na celu znalezienie błędów aplikacji. Liczba znalezionych błędów zależy od:

- Tego, jakie przypadki testowe wchodzi w skład scenariusza (czy są to przypadki niskopoziomowe, czy wysokopoziomowe).
- Tego jak dużo mamy czasu na przeprowadzenie testów.
- Jaki jest cel przeprowadzanych testów (czy mamy potwierdzić, że zmiany dokonane w systemie nie zepsuły działania innych funkcjonalności, czy też mamy zweryfikować, czy nowa funkcjonalność działa tak, jak powinna)?

Scenariusz testowy może również dotyczyć większych funkcjonalności jak, np. testowanie dokonania zakupu w sklepie internetowym przez użytkownika posiadającego konto w sklepie. W tym wypadku przypadki testowe zawarte będą dotyczyć wyszukania przedmiotu, dodania go do koszyka, przejścia do płatności, logowania do aplikacji i na sam koniec płatności za towar.

PRZYPADEK TESTOWY

Przypadek testowy to ciąg instrukcji, jakie należy wykonać w celu zweryfikowania poprawności działania konkretnej funkcjonalności. Przypadki testowe podzielić możemy na niskopoziomowe oraz wysokopoziomowe. Różnią się one między sobą poziomem szczegółowości testów, jakie należy wykonać w celu zweryfikowania przypadku.

W jakim celu piszemy przypadki testowe?

Przypadki testowe piszemy, aby udokumentować w przejrzysty sposób różne możliwości obsłużenia modułów w ramach danej aplikacji. Dobre pokrycie przypadkami testowymi oprogramowania daje nam pewność podczas testów, że nie pominęliśmy żadnej ważnej funkcjonalności. Po zakończeniu testów, na podstawie przypadków testowych możemy budować nasze raporty z wykonanych testów. Dla nowo przyjętych do zespołu ludzi przypadki testowe mogą stać się bardzo dobrym źródłem informacji o niej. Przypadki testowe można również wykorzystać w kontekście testów akceptacyjnych w celu potwierdzenia działania aplikacji zgodnie z oczekiwaniami. Jak widać na powyższych przykładach, przypadki testowe (Test case) są bardzo istotne! Z tego powodu często w trakcie rozmów rekrutacyjnych na testera manualnego pada pytanie właśnie o nie.

Element	Opis
ID	Unikalny numer nadawany automatycznie przy tworzeniu przypadku testowego.
Twórca	Wartość nadawana automatycznie w momencie utworzenia przez użytkownika przypadku testowego.
Tytuł	Krótko zwięźle i na temat co ma być przetestowane.
Cel	Cel wykonania testu.
Warunek początkowy	Najprościej mówiąc, są to dane potrzebne do wykonania testu. Mogą być to zarówno informacje na temat konta użytkownika, jego uprawnień, jak i miejsca w aplikacji, gdzie znajduje się zalogowany użytkownik.
Kroki	Opis co wykonujemy krok po kroku, aby dotrzeć do celu przypadku testowego.
Rezultat	Rezultat wykonania kroku/kroków testowych.
Priorytet	Istnieje możliwość nadawania priorytetów przypadkom testowym, pozwala to w wypadku braku czasu zdefiniować, które testy zostaną wykonane w pierwszej kolejności.
Wykonanie Manualne /Automatyczne	Wartość w polu oznacza, czy dany przypadek ma być wykonywany ręcznie, czy jest pokryty kodem.
Czas na wykonanie	Czas, jaki zgodnie z naszą wiedzą powinien zostać poświęcony na weryfikację przypadku testowego.

Table 1: Elementy przypadku testowego

Przypadek niskopoziomowy

Pisany jest zazwyczaj na potrzeby testów automatycznych i
www.wyszkolewas.com.pl

akceptacyjnych. Podczas jego tworzenia podajemy dokładne dane wejściowe potrzebne do wykonania przypadku. Opisujemy również, jaki jest oczekiwany rezultat wykonania przypadku.

PRZYKŁADOWO:

TYTUŁ: LOGOWANIE DO APLIKACJI ISTNIEJĄCEGO UŻYTKOWNIKA

WARUNEK WSTĘPNY: W SYSTEMIE ISTNIEJE UŻYTKOWNIK USER1 Z HASŁEM HASŁO1, USER1 POSIADA ROLĘ ADMINISTRATORA.

KROKI TESTOWE:







1. URUCHOM APLIKACJĘ.
2. W POLU LOGIN WPISZ „USER1”, W POLU HASŁO WPISZ „HASŁO1”.
3. KLIKNIJ „ZALOGUJ”.

WARUNEK KOŃCOWY: UŻYTKOWNIK USER1 JEST ZALOGOWANY DO APLIKACJI Z UPRAWNIENIAMI ADMINISTRATORA.

Wykonując testy przypadku niskopoziomowego stosujemy się dokładnie do tego, co jest opisane w jego treści.

Przypadek wysokopoziomowy

W przeciwieństwie do przypadku niskopoziomowego pozostawia on testerowi swobodę do testów konkretnej funkcjonalności. W jego treści nie podaje się szczegółowych danych wejściowych ani oczekiwanych rezultatów. Od testera i jego pomysłowości zależy, jak dokładnie przetestuje daną funkcjonalność.

Test Case				
0001-411 : Zalogowanie do panelu klienta - Version 1				
Summary				
Celem testu jest sprawdzenie czy użytkownik jest w stanie zalogować się do panelu klienta.				
Preconditions				
1. Przeglądarka Chrome w najnowszej dostępnej wersji				
2. System Op Windows 10				
3. Konto standardowe użytkownika				
#	Step actions	Expected Results	Execution	
1	1. W polu adresu przeglądarki wpisz adres www.allegro.pl	1. W przeglądarce otworzy się strony główna portalu Allegro	Manual	 
2	Klikamy w przycisk o nazwie "Moje Allegro" w górnym prawym rogu portalu	Rozwinięte zostaje menu	Manual	 
3	Klikamy w pomarańczowy przycisk o nazwie "ZALOGUJ"	Strona się przeladuje wczytując panel logowania użytkownika	Manual	 

ESTYMACJA CZASU PRACY

Każdy projekt ma swój budżet i czas na wykonanie. Ma również zadania (taski), jakie mają być wykonane w ramach projektu. Wykonując estymację czasu pracy, bierzemy pod uwagę wielkość i złożoność zadania, by podać czas, jaki według nas będzie potrzebny na wykonanie go. Tester pytany o estymację czasochłonności będzie wyceniał czas potrzebny na wykonanie testów funkcjonalności, napisanie dokumentacji.

Dane potrzebne by dobrze wycenić zadanie

Estymacja czasu pracy nie powinna być spontaniczna, ale przemyślana na podstawie konkretnych danych informacji, jakie zbierzemy o zadaniu.

Dlatego, zanim na pytanie „Ile potrzebujesz czasu, by przetestować logowanie?” odpowiesz „Trzy godziny” odpowiedz sobie na następujące pytania.

Zespół

Kto jest w zespole, w którym pracujesz? Czy stanowią go ludzie, którzy pracują w projekcie od dłuższego czasu czy też ludzie nowi?

Założmy, że pracujesz w zespole złożonym z doświadczonych i pracujących w projekcie od jakiegoś czasu osób. Wówczas nakład pracy potrzebnej do wykonania zadania będzie mniejszy niż w przypadku zespołu, w którym jest kilka osób z krótkim stażem pracy w projekcie. Wynika to z tego, że osoby nieznające produktu będą potrzebować czas na wdrożenie się w niego. Dlatego estymując czas testów funkcjonalności, weź pod uwagę kto będzie poprawiał błędy, które zgłosisz.

Jak duże jest zadanie?

Duże zmiany w aplikacji to też odpowiednio długi czas poświęcony na ich wdrożenie. To też zwykle duża ilość dokumentacji do przeczytania oraz do napisania. Testowanie większej funkcjonalności wiąże się z tym, że testujemy ją w mniejszych „paczkach”. W związku z tym programista stale ingeruje w kod testowanego komponentu dodając nowe funkcjonalności („paczki”). Co za tym idzie nawet przetestowany wcześniej obszar może się zepsuć. A to z kolei powoduje, że pod koniec developmentu powinniśmy przeprowadzić dodatkowe Smoke testy. Tak więc estymując czas testów dużej funkcjonalności powinniśmy wziąć pod uwagę testy „paczek” i całej funkcjonalności.

Małe zadania mają to do siebie, że łatwo je oszacować ze względu na ich wielkość i przeważnie małe skomplikowanie. Nie chodzi mi o skomplikowanie w kodzie, ponieważ nawet najmniejsze zmiany mogą skutkować testami Regresji. Ale o to, że najczęściej otrzymujemy małą ilość „paczek” do testów. Mamy też dużo mniej dokumentacji do przetworzenia, by przetestować funkcjonalność.

Czy aplikacja, której testy dotyczą ma stabilny kod/środowisko?

Im większa stabilność kodu, tym mniejsza szansa na dużą ilość błędów w modułach, które już rozwijaliśmy od jakiegoś czasu. Łatwiej jest też wprowadzać kolejne moduły do aplikacji zbudowanej na „dobrym” kodzie. To samo tyczy się środowiska, na którym mamy postawioną aplikację. Im jest ono bardziej zadbane, tym mniej czasu stracimy, by dało się aplikację testować.

Czy zadanie dotyczy samych testów?

Czy poza testami masz wziąć pod uwagę pisanie / aktualizację dokumentacji testowej, planowanie testów i retesty?

Bardzo ważną rzeczą podczas szacowania jest pamiętanie o tym, że estymacja czasu pracy na zadanie to nie same testy. Pamiętaj, że poza nimi powinienś wycenić również czas potrzebny na wykonanie dodatkowych prac. Należą do nich zawsze planowanie, pisanie dokumentacji testowej, zgłoszenie różnic pomiędzy specyfikacją a aplikacją oraz retestowanie defektów.

Czy istnieje zależność zespołowa?

Czy do wykonania zadania konieczna jest współpraca z innymi zespołami (zwłaszcza z zespołami związanymi z inną firmą)?

Bardzo często nowo tworzona funkcjonalność ma zintegrować się za pomocą usług z innymi systemami. Systemy te niekoniecznie muszą być tworzone przez firmę, w której pracujemy. To stwarza dodatkowe ryzyko w kontekście estymacji czasu pracy. W takich sytuacjach nie zależy od nas to ile potrwa wyjaśnianie problemów czy naprawa błędów po stronie dostawcy usługi.

Jaka jest twoja wiedza o produkcie, który będzie testowany?

Estymacja powinna być wykonywana przez osobę, która realnie będzie

wykonywała testy. Przykładowo, jeżeli trafimy do nowego projektu i z marszu będziemy mieli zacząć testować nowe funkcjonalności, to należy doliczyć konieczność głębszego wejścia w dokumentację analityczną – chyba, że założyliśmy testy za pomocą technik opartych na doświadczeniu.

Jaki jest czas, jaki pozostał do oddania projektu?

Czas jest kolejnym niedocenianym czynnikiem w procesie estymacji. Bardzo często projekty sterowane są datą wdrożenia oprogramowania na produkcję. Niestety w takich projektach zdarza się, że w związku z ilością zadań do projektu są dorzucane nowe osoby, które mają pomóc szybciej wykonać zadanie. I teraz zależy, kiedy są one przyjmowane. Jeśli jest to na początku projektu, wtedy estymując czas pracy powinienś dodać godziny potrzebne tym osobom na wdrożenie. Czas ten zostanie jednak zrekompensowany tym, że te osoby odpowiednio długo będą później pracować w projekcie. Ale jeśli nowe osoby są dodawane do projektu tuż przed wdrożeniem, to często czas potrzebny na ich wdrożenie i pomoc udzielaną przez doświadczonych pracowników powoduje, że prace nad projektem zwalniają, zamiast przyspieszać.

Czy pracujesz w tym samym czasie w jednym, czy w kilku projektach?

Pracując w kilku projektach, podczas estymacji weź pod uwagę, czas potrzebny na przełączenie się między zadaniami/projektami. Przykładowo mając dwutygodniową przerwę w pracy w projekcie, może być konieczne przeczytanie dokumentacji.

Estymacja czasu pracy powinna być zawsze uczciwa

Przy okazji estymacji czasu pracy musimy dbać o to, żeby była ona zawsze uczciwa. Głównym tego powodem jest zaufanie, jakie budujemy sobie poprzez sumienną i szczerą estymację. Daje ona klientowi jasny przekaz, że „jesteśmy w stanie oddać x zadań o standardach jakościowych, które wcześniej ustaliliśmy”. Dodatkowo nam samym daje ona możliwość lepszego poukładania zadań z perspektywy najbliższych tygodni.

Jaki projekt jest łatwiej wyestymować Nowy czy taki, który jest realizowany od jakiegoś czasu?

Moim zdaniem dużo łatwiej jest wyestymować czas pracy dla produktu, który jest już na rynku. Głównym powodem, tego jest możliwość porównywania czasów, jakie szacowałem na przestrzeni ostatnich x miesięcy dla tego produktu.

Zaletą nowo robionego projektu, może być za to świeży start, który nie jest możliwy przy projekcie trwającym już kilka miesięcy. Dodatkowo w przypadku trwającego projektu przeważnie jesteśmy proszeni o estymację funkcjonalności aplikacji. Natomiast dla nowo powstającej aplikacji będzie to określenie czasochłonności stworzenia nowego systemu.

Nasze wyceny nie zawsze będą perfekcyjne, natomiast powinniśmy dążyć do tego, aby były one jak najbliżej ideału. Biorąc wymienione w tym artykule czynniki, możemy się domyślić, że proces wytwarzania oprogramowania nie jest rzeczą prostą tylko składa się na niego wiele czynników, niektórych oczywistych i jasnych a innych ukrytych i straszących nas w najmniej spodziewanym momencie. Dlatego też uważam, że należy zbierać sobie dane z wycen i próbować przeanalizować ich wyniki. Dla mnie osobiście dobrą estymatą jest próg błędu rzędu +/- pół dnia pracy na zadaniu, które ma trwać przez 3 tygodnie (chodzi o testy). Jeżeli udaje mi się uzyskać tego typu wyniki. to zapisuję sobie składowe oraz czynniki poboczne, jakie wpłynęły na moją wycenę.

Techniki estymacji czasu pracy

- Estymacja na podstawie danych historycznych.
- Estymacja czasu pracy „bottom-up”.
- Ocena ekspercka.
- Trzystopniowa estymacja.
- Planning poker.
- Wycena procentowa.
- Wycena oparta na ryzykach.

Estymacja na podstawie danych historycznych

Dosyć prosty sposób estymacji, ponieważ w ramach tej techniki porównujemy projekty, które wcześniej estymowaliśmy a są identyczne jak projekt, który mamy dostarczyć. Uwaga w ramach tej techniki istnieje ryzyko odnośnie pomyłki co do podobieństwa aplikacji uruchomionej przez nas wcześniej a aplikacji dopiero przez nas tworzonej. Skutkujące złym oszacowanie czasu potrzebnego na wykonanie testów.

Estymacja czasu pracy „bottom-up”

Cechuje ją bardzo duża dokładność, jeżeli chodzi o wyniki, natomiast minusem tej techniki jest konieczność poświęcenie dużej ilości czasu, żeby ją wykonać. Dodatkowo warto wiedzieć, że ta technika nie zawsze będzie dla nas dostępnym narzędziem, np. w wypadku braku dokumentacji analitycznej nie będziemy w stanie wykonać tego typu estymacji.

Ocena ekspercka

Technika oparta na doświadczeniu osoby, która wycenia czas potrzebny do wykonania zadania.

Planning poker

Jest to technika typowa dla projektów Scrumowych. Różni się ona znacząco od pozostałych wymienionych, ponieważ przy poprzednich technikach przeważnie estymujemy czas w Godzinach lub „Man day” (osobo dniach). Natomiast w planing pokerze estymujemy za pomocą Story Pointów. Story Point jest jednostką złożoności lub ryzyka historyjki. W przeciwieństwie do poprzednich technik tutaj każde zadanie jest estymowane przez cały zespół. Skutkuje to tym, że tester bierze czynny udział w estymacji czasu programisty, jak i programista estymuje czas, jaki będzie potrzebny na testy. Wygląda to tak, że każdy członek zespołu szacuje czas wykonania każdego zadania.

Typowa skala wartości jakie podczas głosowania można podać to: 1/2, 0, 1, 2, 3, 5, 8, 13, 21

W tej skali wartości minimalne typu 1 lub 2 to zadania banalne do wykonania, z którymi bez problemu sobie poradzimy jako zespół. Wartość 21 jest specyficzna oznacza ona, że nie jesteśmy w stanie wykonać zadanie w ramach jednego cyklu, więc trzeba je rozbić na mniejsze zadania, a następnie ponownie dokonać estymacji.

Przykładowo:

Mamy zespół składający się z ośmiu osób. Podczas estymacji czasu wykonania zadania podały one następujące wartości:

Pięć osób zagłosowało za wartością 5, dwie osoby zagłosowały za wartością 2, natomiast jedna za 21.

W takim wypadku osoby, które dały skrajne wartości tłumaczą się, dlaczego dokonały takiej estymacji, a następnie estymujemy ponownie.. aż otrzymamy podobne wyniki w skali zespołu.

Plusem tej techniki jest aktywność całego zespołu przy estymacji.

Co w przypadku testera może pomóc w łatwiejszym zrozumieniu aplikacji, złożoności zadania oraz potencjalnych ryzyk wynikających z przeprowadzania testów danej funkcjonalności. Minusem jest tak zwana bezpieczna piątka, jest to efekt konieczności tłumaczenia się z rozbieżności w wycenie pomiędzy osobami (skrajne wartości, o których wcześniej wspomniałem). Dlatego część osób, może wychodzić z założenia, że lepiej dać jakąś bezpieczną wartość, żeby nikt się nas nie pytał, dlaczego tak, a nie inaczej.

Wycena procentowa

Sposób estymowania czasu pracy na podstawie czasu, jaki przedstawili developerzy na jej wykonanie. Przeważnie jest to wartość 20%-30% czasu, jaki muszą programiści poświęcić na zrobienie funkcjonalności.

Plusy

Estymacja zajmuje kilka minut.

Minusy

Czasami zadania realizowane przez programistów generują nam więcej pracy, niż oni sami na nie musieli poświęcić. W takich wypadkach po wykonaniu estymacji metodą wyceny procentowej nie będziemy w stanie wykonać w wyznaczonym (przez siebie samych) czasie zadań.

Wycena oparta na ryzykach

Sposób estymowania czasu pracy polegający na dogłębnym

przeanalizowaniu zadań i wszystkich możliwych przeciwności, jakie mogą wystąpić. Przy tego typu wycenie warto brać pod uwagę czynniki takie jak:

- Czas na zgłoszenie błędów + retesty.

- Czas na przygotowanie dokumentacji testowej.

- Czas na pielęgnowanie środowiska oraz inne czynności związane z tym zadaniem.

Plusy tej metody

Dużo precyzyjnie określony czas, jaki jest potrzebny na wykonanie zadań. W tej metodzie rzadko występują pomyłki dotyczące czasochłonności. Dzięki tej wycenie tester przy okazji poznaje dużo lepiej funkcjonalności, jakie mają zostać dodane w ramach cyklu.

Minusy

Potrzeba większej ilości czasu na dokonanie estymacji czasu pracy.

Co zrobić, jeśli podana przez Ciebie estymacja czasu pracy nie została zaakceptowana?

Ja w takim przypadku siadam ponownie nad danymi, które zebrałem w celu estymacji i analizuje je. Robię to w celu wyeliminowania lub zmniejszenia nakładu czasu dla zadań o niskim priorytecie. Następnie przedstawiam propozycję estymacji czasu wraz z informacją, co zostało zmienione w stosunku do poprzedniej.

USPRAWNIENIA W PRACY TESTERA

Code review.

Polega na tym, że programiści są zobowiązani sprawdzić sobie wzajemnie kod w celu wyeliminowania błędów aplikacji. W trakcie Code review można uruchomić testy automatyczne, by wyłapać błędy aplikacji, zanim przekaże się ją do testów.

Testy automatyczne.

Testy automatyczne mogą bardzo pomóc w utrzymaniu jakości w aplikacji na zaplanowanym poziomie. Celem tworzenia testów automatycznych nie jest znalezienie błędów, ale utwierdzenie się w przekonaniu, że ich nie ma w najważniejszych ścieżkach aplikacji. Nie jest możliwe zautomatyzowanie testów całej aplikacji i każdej funkcjonalności. Nie robi się tego, ponieważ testy automatyczne są kosztowne przy tworzeniu i utrzymaniu. Zwykle powinno się automatyzować te przypadki testowe, w których nie ma planowanych zmian w kodzie aplikacji. Co za tym idzie w przypadku raz dobrze napisanych testów automatycznych, można zakładać, że powinny one działać cały czas bez błędów i utwierdzać, co do poziomu jakości aplikacji.

Ustalenie norm jakości.

Powinniśmy przed rozpoczęciem tworzenia oprogramowania ustalić sobie normy jakości, np. nie może być ani jedno otwarte zgłoszenie o priorytecie „Wysoki” na zakończenie projektu.

TEORIĘ JUŻ ZNASZ, CO DALEJ

Poznałeś już teorię testowania oprogramowania. Czas zacząć (o ile już tego nie zrobiłeś) ćwiczyć zgłaszanie błędów, pisanie przypadków, scenariuszy, znajdowanie błędów.

Kolejnym krokiem jest przygotowanie się do rekrutacji. Napisz CV i list motywacyjny, w którym wyróżnij to co ważne jest na stanowisku testera oprogramowania. Utwórz i uzupełnij danymi profil na LinkedIn.

Przygotuj się do rozmowy rekrutacyjnej. Uważam, że przygotowanie do pytań miękkich jest równie ważne jak to do pytań z wiedzy. Dlatego ćwicz na głos odpowiadanie na nie w domu przed lustrem lub do przyjaciela. Naucz się odpowiadać na najczęściej padające na rozmowach pytania z teorii.

Jak widzisz teoria jest bardzo ważna, ale ważna jest również praktyka oraz przygotowanie się do rozmowy rekrutacyjnej. Kompleksowy proces przebranżowienia zawierać powinien wszystkie te elementy.

Lista najczęściej padających pytań na rozmowie na testera oprogramowania

Dość często na grupach na Facebook pojawiają się pytania o to jakie pytania padają na rozmowach rekrutacyjnych. Poniższa lista to odpowiedzi jakie padają na takie posty w komentarzach.

Pytania dotyczące testowania

1. Co zrobisz w sytuacji, gdy programista zamknie błąd ze statusem “as designed”?
2. Jakie są narzędzia używane przez testerów?
3. Dlaczego chcesz być testerem?
4. Ile chcesz zarabiać?
5. Jaka jest rola testera w zespole?
6. Czy tester może być aktywny w zespole na etapie tworzenia dokumentacji projektowej?
7. Po co testujemy oprogramowanie?
8. Jakie cechy idealnego testera posiadasz?
9. Jak przetestujesz kalkulator, krzesło, ołówek, parasolkę?
10. Czemu studzienka kanalizacyjna jest okrągła?
11. Czemu tory kolejowe są budowane na wałach?
12. Jak byś określił wagę samolotu mając do dyspozycji aparat fotograficzny?
13. Ile cegieł potrzeba do zbudowania domu?
14. Ołówek i gumka kosztują 1.10zł, ołówek jest o 1zł droższy. Ile kosztuje gumka?
15. Co to są testy funkcjonalne i niefunkcjonalne?
16. Różnica między testami czarnoskrzynkowymi i białoskrzynkowymi.
17. Czym są testy jednostkowe?
18. Czym są testy regresyjne? Do czego służą?
19. Na czym polegają retesty?
20. Jak powinien wyglądać przypadek testowy?
21. Co powinno być w zgłoszeniu błędu?
22. Cykl życia defektu
23. Pytania z zakresu ISTQB i o zastosowanie wiedzy z ISTQB w praktyce.
24. Co to jest piramida testowania?
25. Jakie są rodzaje testów? Scharakteryzowanie ich.
26. Omówienie priorytetów błędów i podanie przykładów błędów i ich priorytetów.

27. Porównanie testów regresji z retestami.
28. Czym są przypadki graniczne?
29. Bug priority vs. bug severity
30. Czym jest i na czym polega testowanie w chmurze (cloud testing)?
31. Zwykle rozwiązywane na kartce papieru. Sprawdzają teorię w praktyce.
32. Wydrukowana tabela z danymi. Przy pomocy SQL-a wyszukaj konkretne dane i posortuj.
33. Tablica decyzyjna i wartości brzegowe.
34. Napisz scenariusz testowy.
35. Zgłoś błąd.
36. Napisz testy pozytywne i negatywne dla formatki.
37. Algorytm quick sort
38. Wydrukowany formularz na kartce – opowiedz jak byś go przetestował.

Pytania miękkie

1. Jak przekonałbyś kogoś do swojej racji? Jakie będą twoje argumenty?
2. Jak pracowałbyś z trudnym zespołem / osobą?
3. Co chciałbyś robić za 5 lat? Gdzie widzisz siebie za 5 lat?
4. Co zrobisz w stresującej sytuacji? (Sytuacja jest podana)
5. Jaki film ostatnio oglądałeś?
6. Jakim typem osoby byłeś na studiach? (Ktoś, kto robi wszystkie notatki dla każdego, ktoś, kto jest zawsze przygotowany na każde zajęcia, ktoś, kto robi wszystko w ostatniej chwili.)
7. Jaka jest twoja strategia osiągnięcia celu?
8. Dlaczego chcesz pracować w tej firmie?
9. Dlaczego ta firma ma cię zatrudnić? Co w twojej osobowości sprawia, że dobrze jest mieć cię w zespole?
10. Jak ludzie cię widzą? Co mówią o tobie? Co twój przyjaciel mógłby nam powiedzieć o tobie, jeśli on / ona musi cię opisać?
11. Jaka była twoja ostatnia porażka, co to było i czego się nauczyłeś?
12. Co robisz po pracy? Jakie jest twoje hobby?

Ćwicz testowanie

Zacznij od zrobienia zadań z Mr Buggy. Znalezione w nim błędy staraj się opisywać zgodnie z zasadami opisywania błędów. Jeśli nie masz zainstalowanego programu do zgłaszania błędów, użyj kartkę i długopis. Na końcu tego ebooka znajdziesz formularze zawierające wszystkie te pola. Możesz wydrukować te strony i przećwiczyć na nich.

Dobrym sposobem na zdobywanie praktyki jest testowanie na uTest, czy innej platformie crawdtesting. Doświadczenie to możesz wpisać do swojego CV.

Napisz przypadki testowe (nisko i wysokopoziomowe) dla wybranych funkcjonalności w telefonie. Jeśli nie masz zainstalowanego programu, napisz te przypadki na kartce. Wypełnij każde pole jakie zwykle uzupełnia się w programie.

Wyznacz przypadki testowe na podstawie wartości brzegowych czy tablic decyzyjnych.

Na kolejnej stronie znajdziesz listę tego, co moim zdaniem trzeba zrobić, by maksymalnie przygotować się do zawodu testera.

Lista platform crawdtestingowych

- <http://www.utest.com/> | <http://www.applause.com/>
- <http://whatusersdo.com/>
- <http://trymyui.com/>
- <http://www.testbirds.com/>
- <http://www.usertesting.com/>
- <https://crowdsourcedtesting.com/>
- <https://www.pay4bugs.com/>
- <https://mycrowd.com/>
- <https://test.io/>
- <https://usabilityhub.com/>
- <https://globalapptesting.com/>
- <https://www.bugfinders.com/>
- <https://we-are-testers.com/>



Mój poziom przygotowania do zawodu testera oprogramowania

- ☐ Mam utworzony i uzupełniony aktualnymi danymi profil na LinkedIn
- ☐ Mam napisane aktualne CV
- ☐ Mam napisany list motywacyjny
- ☐ Umiem obsługiwać Jira, Mantis lub inny program do zgłaszania błędów
- ☐ Orientuję się w TetLink - umiem utworzyć przypadek testowy, scenariusz testowy, plan testów i wygenerować raport z testów
- ☐ Znam narzędzie do robienia zrzutów i umiem zrobić zrzut
- ☐ Umiem opisać błąd
- ☐ Umiem napisać przypadek testowy
- ☐ Umiem napisać scenariusz testowy
- ☐ Znam podstawy HTML-a, CSS-a
- ☐ Znam podstawy SQL-a
- ☐ Znam odpowiedzi na najczęściej padające pytania na rozmowach rekrutacyjnych
- ☐ Zarejestrowałem się na uTest i aktywnie uczestniczę w testach
- ☐ Przetestowałem przynajmniej 2 strony i znalazłem na nich błędy

Teorię już znasz, co dalej

TYTUŁ

Krótko zwięźle i na temat co ma być przetestowane.

CEL

Cel wykonania testu.

**WARUNEK
POCZĄTKOWY**

Dane potrzebne do wykonania testu. Mogą być to zarówno informacje na temat konta użytkownika, jego uprawnień, jak i miejsca w aplikacji, gdzie znajduje się zalogowany użytkownik.

KROKI	REZULTAT
1.	1.
2.	2.
3.	3.
4.	4.
5.	5.
6.	6.
7.	7.
8.	8.
9.	9.
10.	10.
11.	11.
12.	12.
13.	13.
14.	14.

Opis co wykonujemy krok po kroku, aby dotrzeć do celu przypadku testowego.

Rezultat wykonania kroku/kroków testowych.

PRIORYTET

Wysoki, Średni, Niski.

CZAS NA

WYKONANIE Czas potrzebny na wykonanie testów.

www.wyszkolewas.com.pl

www.wyszkolewas.com.pl

TEMAT

Krótki opis mówiący co nie działa w aplikacji. Tytuł powinien w sposób jasny i czytelny mówić co nie działa w aplikacji.

PRIORYTET

POWTARZALNOŚĆ

Wysoki, Średni, Niski.

Ile prób odtworzenia udaje się powtórzyć

OPIS

Dane opisowe błędu. Mogą to być informacje szczegółowe na temat tego, jaki komunikat się wyświetlił. Albo dodatkowe informacje początkowe, jakie muszą zaistnieć.

KROKI TESTOWE

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

11.

12.

13.

14.

Krok po kroku, tak jakbyśmy tłumaczyli komuś, kto nigdy aplikacji na oczy nie widział, co trzeba zrobić, by błąd odtworzyć.

ŚRODOWISKO

Nazwa i wersja przeglądarki, wersja aplikacji, na której wystąpił błąd.