PASJA TESTOWANIA

KSIĄŻKA DLA KAŻDEGO, KTO CHCE ZOSTAĆ TESTEREM

BEZPŁATNY FRAGMENT

Krzysztof Jadczyk

Wydanie pierwsze
ISBN 978-83-955930-0-0

Korekta – Małgorzata Woźna
Wydawca – Bugfree Krzysztof Jadczyk, Wrocław 2019

Wprowadzenie

Od Autora
Wstęp

Wprowadzenie

Od autora

Jak to się stało, że napisałem tę książkę? Przecież nie było to moim marzeniem z dzieciństwa. Najpierw pojawił się pomysł pisania artykułów, które publikuję od września 2018 roku na LinkedIn'ie, a także na Facebooku. Od czerwca 2019 moje wpisy znajdziesz na bugfreeblog.com. Tematyka jest różna, choć zdecydowanie przeważa testowanie oprogramowania, które stało się moją pasją jakieś 13 lat temu. To wtedy zacząłem stawiać pierwsze kroki w zawodzie. Po napisaniu ponad trzydziestu artykułów postanowiłem zabrać się za większy projekt. Tak narodził się pomysł napisania tej książki. Ten, kto miał okazję śledzić moje poczynania na tych portalach społecznościowych, znajdzie tu treści, które już wcześniej ujrzały światło dzienne. Część z nich jest mniej lub bardziej przeredagowana, by tworzyć spójną całość również z nowymi, wcześniej niepublikowanymi fragmentami.

Książka jest kontynuacją pewnej myśli, która towarzyszyła mi podczas tworzenia poszczególnych wpisów. Zamysł był taki, by opisać zagadnienia związane z pracą testera dla osób, które rozpoczynają przygodę z testowaniem. Książka powstała z mojej pasji testowania, by pobudzić tę pasję u innych.

Obserwując grupy tematyczne związane z testowaniem w mediach społecznościowych, widzę, że coraz więcej osób chce rozwijać się w kierunku testowania lub chociaż zobaczyć, czy się w tym odnajdą. Słyszałem mnóstwo pytań o to, jak zacząć, czego się uczyć, z jakich materiałów korzystać i zrozumiałem, że być może wciąż zbyt mało jest przydatnych treści. Dlatego teraz chciałbym oddać Ci, drogi Czytelniku, tę oto książkę. Mam nadzieję, że znajdziesz w niej wartościowe treści.

Zanim jednak przejdziesz do kolejnych rozdziałów, chciałbym, żebyś wiedział, że treści zawarte na tych stronach są efektem mojego doświadczenia i przedstawiają mój punkt widzenia. Nie znajdziesz tu uniwersalnych zasad do zastosowania w każdej sytuacji. Zresztą tak właśnie jest z samym testowaniem. W swojej przyszłej pracy jako tester możesz nie znaleźć jednej poprawnej odpowiedzi na rozwiązanie pewnego zagadnienia. Najczęściej będzie to sytuacja typu: TO ZALEŻY. Część rozdziałów jest tylko zajawką pewnych tematów do dalszego zgłębienia. Celowo nie będę ich rozwijał ze względu na to, że bez problemu znajdziesz w sieci tutoriale lub inne materiały. Nie jest to jednak znak, że dany temat nie jest wart tego, aby poświęcić mu więcej czasu. Zdecydowanie wszystko, co jest zawarte w tej książce, uważam za minimum, by myśleć o rozwoju w kierunku testera oprogramowania. Nie rzucaj się od razu na naukę automatów i języków programowania. Najpierw zbuduj fundamenty do dalszego rozwoju, którymi według mnie jest odpowiedni sposób myślenia. Uważam, że dopiero wtedy jest sens iść dalej. Pamiętaj – najpierw solidne fundamenty, a później stawianie murów wiedzy, na których kolejno będą utrzymywać się coraz większe osiągnięcia.

Wstęp

Ważne jeszcze jest to, żebyś odpowiedział sobie na jedno pytanie. Brzmi ono następująco: Dlaczego chcę zostać testerem? Musisz wiedzieć, że droga, jaką obierasz, nie jest łatwa. Samo przygotowanie, by móc pójść na pierwszą rekrutację, zajmie pewnie kilka miesięcy. I to nie koniec. To będzie dopiero początek nieustannego uczenia się w późniejszych miesiącach i latach pracy. Czy zatem warto iść w tym kierunku? Być może odpowiedź znajdziesz właśnie w tej książce.

Zastanów się też, w jaki sposób chcesz tę drogę pokonać i przygotuj konkretny plan nauki. Plan, który jak kompas będzie wskazywał kierunek wędrówki. Bez niego możesz błądzić i momentami kręcić się bez celu, tracąc cenny czas i motywację. Nie zapomnij też celebrować nawet małych sukcesów! Warto o tym pamiętać. Wiem to z własnego doświadczenia. To będzie niezwykle ważne i pomoże Ci przetrwać chwile zwątpienia, które mogą się pojawiać.

Wyznacz sobie plan i działaj. Nie ma oczywiście nic złego w pojawianiu się odstępstw od planu. To pewnie zdarzy się nie raz i może wynikać z rosnącej wiedzy. Wtedy zmodyfikuj plan i dostosuj go do zmieniających się warunków. Ważne, żeby ćwiczyć i iść w odpowiednim kierunku.

Podstawy testowania

Czym jest testowanie?
Przypadki testowe
Zgłoszenie defektu
Cykl życia defektu

Podstawy testowania

Czym jest testowanie?

Testowanie oprogramowania jest procesem związanym z wytwarzaniem oprogramowania. Polega na weryfikacji poprawności działania aplikacji. Może być przeprowadzone w oparciu o specyfikację i powinno sprawdzać zgodność systemu z wymaganiami użytkowników. Testowanie jest jednym z procesów zapewnienia jakości. Musisz wiedzieć, że testowanie samo w sobie nie podnosi jakości oprogramowania. Służy ocenie jakości, a także budowaniu zaufania do weryfikowanego produktu. Jakość natomiast może zostać podniesiona poprzez wprowadzenie poprawek i wyeliminowanie znalezionych błędów.

Proces testowania aplikacji może skupiać się na weryfikacji zaimplementowanych funkcjonalności, które zostały wytworzone w oparciu o wymagania użytkownika. Mówimy wtedy o testach funkcjonalnych. Często są też sprawdzane pewne cechy jakościowe systemu, które nie są związane z konkretną funkcjonalnością, jak chociażby bezpieczeństwo czy wydajność. W takim wypadku mówimy o testach niefunkcjonalnych.

Przypadki testowe (Test Cases)

Przypadki testowe są wciąż najczęściej przygotowywanym przez testerów rodzajem dokumentacji. Z tego powodu dość często możesz trafić na pytania dotyczące tego zagadnienia podczas rozmowy rekrutacyjnej. Warto zatem zgłębić ten temat. Jak zatem taki przypadek testowy powinien wyglądać i jakie elementy zawierać?

Zanim przejdziemy do przykładu, zajrzyjmy najpierw do definicji zaczerpniętej ze „Słownika wyrażeń związanych z testowaniem” SJSI.

Przypadek testowy to zbiór danych wejściowych, wstępnych warunków wykonania, oczekiwanych rezultatów i końcowych warunków wykonania opracowany w określonym celu lub dla warunku testowego, jak wykonanie pewnej ścieżki programu lub zweryfikowanie zgodności z konkretnym wymaganiem.

Przykładowy przypadek testowy

Na potrzeby tego rozdziału oraz kolejnego, dotyczącego defektów, posłużę się zadaniem #20 z Mr Buggy 3. Przygotuję do niego przykładowy przypadek testowy, żeby zademonstrować, jak może wyglądać. Zgodnie z ideą tej książki, nie jest to jedyne słuszne podejście, bo takie nie istnieje. Mimo tego poniższy przykład powinien dać Ci pewien pogląd na to zagadnienie.

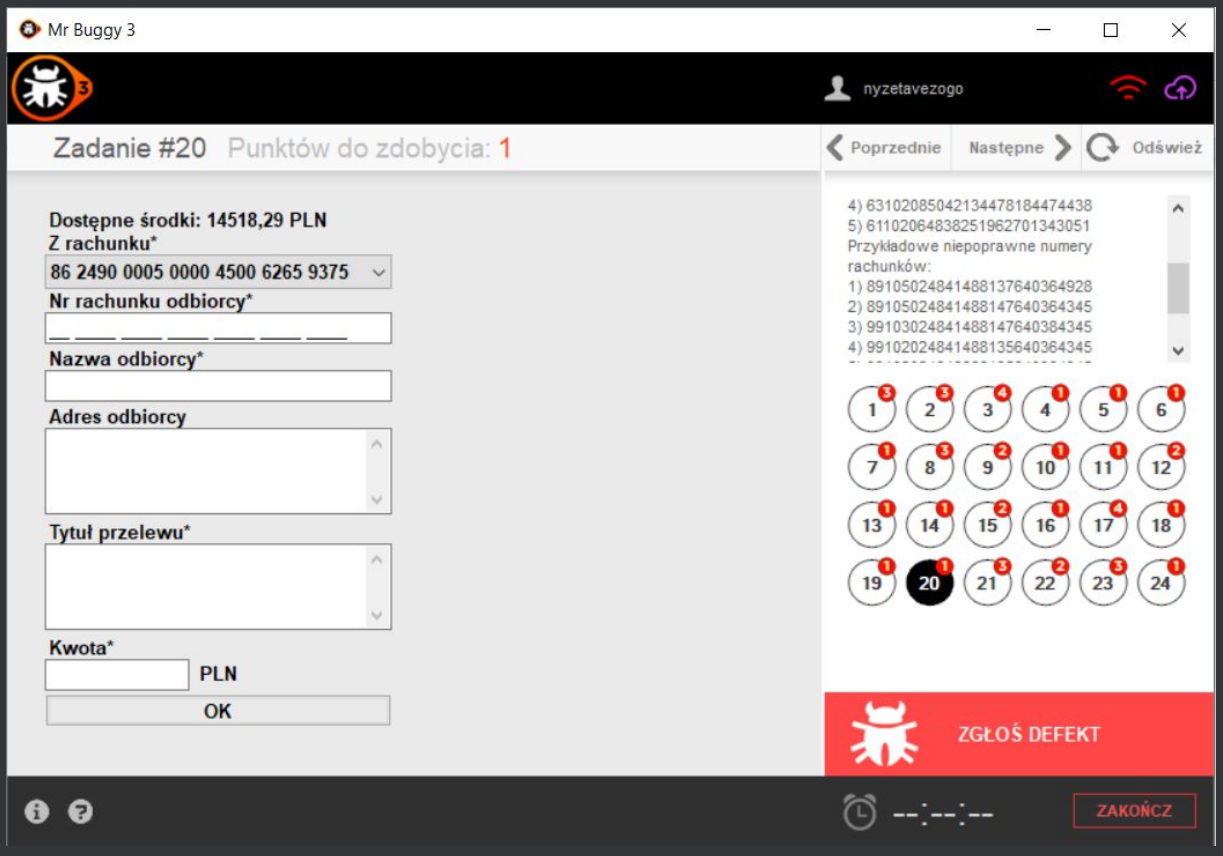
Treść zadania:
Zadanie przedstawia funkcję wysyłania przelewów w internetowym koncie bankowym. Odszukaj i zgłoś defekt.

Przykładowe poprawne numery rachunków:
1) 83109069928729691303181767
2) 97116052168159436723349207
3) 33102022251281637642546453
4) 63102085042134478184474438
5) 61102064838251962701343051.

Przykładowe niepoprawne numery rachunków:
1) 89105024841488137640364928
2) 89105024841488147640364345
3) 99103024841488147640384345
4) 99102024841488135640364345
5) 99102024846666135640364345.

Pola oznaczone gwiazdką są wymagane. Rozłożenie pól i przycisków jest zgodne z wymaganiami klienta.

Spójrz jeszcze na to, jak wygląda formularz.



Przypadek testowy do powyższego zadania może wyglądać jak poniżej.

Title/no	Buggy3_001	
Description	Sprawdź czy można wysłać przelew z ujemną wartością w polu *Kwota*	
Precondition	Istniejące konto z dodatnim saldem	
Users/roles	Klient banku, właściciel konta	
Test data	Przykładowe poprawne numery rachunków: 1) 83109069928729691303181767 2) 97116052168159436723349207 3) 33102022251281637642546453 4) 63102085042134478184474438 5) 61102064838251962701343051	
Step	**Action**	**Expected result**
1	Wprowadź poprawny numer konta w pole *Nr rachunku odbiorcy*	Pole *Nr rachunku odbiorcy* uzupełnione
2	Wypełnij pole *Nazwa odbiorcy* dowolną wartością	Pole *Nazwa odbiorcy* uzupełnione
3	Wypełnij pole *Tytuł przelewu* dowolną wartością	Pole *Tytuł przelewu* uzupełnione
4	Wprowadź ujemną wartość w pole *Kwota* i naciśnij przycisk OK	Brak możliwości wprowadzenia wartości ujemnej lub komunikat walidacyjny po wciśnięciu przycisku OK

Poza tym, co zawarte jest w powyższej tabeli, mogą też pojawić się dodatkowe elementy. Jednym z nich jest priorytet, którym można określić, które z testów powinny zostać wykonane

w pierwszej kolejności. To może się przydać, np. kiedy brakuje czasu na wykonanie wszystkich zaplanowanych wcześniej testów. Wtedy zyskujemy możliwość wybrania zestawu najistotniejszych przypadków i pominięcia tych najmniej krytycznych (z niższym priorytetem).

W trakcie wykonywania testów mogą pojawić się także informacje związane ze środowiskiem (np. TEST, UAT, SIT), na którym przypadek został wykonany. Istotne będzie też to, na jakiej wersji aplikacji test powinien zostać wykonany oraz na jakiej konfiguracji, np. na której przeglądarce, systemie operacyjnym czy rozdzielczości ekranu.

Warto też wiedzieć, że jest sporo narzędzi, które można wykorzystać jako repozytorium testów. To w nich tworzone i przechowywane są przypadki testowe, które później czekają na ich wykonanie. Więcej informacji o narzędziach znajdziesz w rozdziale „Narzędziownia".

Egzekucja testów

Egzekucja testów (test execution) polega na wykonaniu kroków zdefiniowanych w test case, a następnie nadaniu odpowiedniego statusu w narzędziu. Status zależy od otrzymanego wyniku testu. Najczęściej spotykane statusy to:

- **to do** – test został zaplanowany i czeka na wykonanie
- **executing/in progress** – oznacza, że dany przypadek jest w trakcie wykonywania
- **passed** – w momencie kiedy test przechodzi bez problemu
- **failed** – gdy rezultat aktualny jest inny niż oczekiwany. W takim przypadku powinien zostać zgłoszony błąd. Zgłoszenie defektu zostało opisane w kolejnym podrozdziale. Większość narzędzi pozwala na powiązanie defektu z test casem, w trakcie wykonania którego został wykryty
- **blocked** – status używany w przypadku, gdy wykonanie danego testu jest niemożliwe w danym momencie.

Odpowiednie oznaczenie statusów pozwala na śledzenie postępu prac. Na bazie zgromadzonych w ten sposób danych można też budować różnego rodzaju metryki (np. przypadki testowe wg statusu wykonania, przypadki testowe wg priorytetu), wizualizacje. To pozwala lepiej zobrazować status wykonania testów oraz ułatwia przekazanie informacji na temat jakości osobom zainteresowanym. Przykładowe metryki zostały opisane w oddzielnym rozdziale.

Po co komu przypadki testowe?

Wiesz już, jak może wyglądać test case. Teraz czas na małą dygresję, która, mam nadzieję, rzuci trochę więcej światła na podejście do ich tworzenia. Czy zatem podczas pracy w projekcie trzeba pisać przypadki testowe?

Cytując Maxa, bohatera jednego z moich ulubionych filmów „E=mc^2", mogę stwierdzić, że:

bo to zależy. Ci, którzy mają już trochę doświadczenia, zapewne wiedzą, o czym mówię. Wpływ na to ma wiele czynników (wymienionych poniżej) i kontekst, w jakim przyjdzie Ci pracować. To samo tyczyć się może innych dokumentów testerskich, takich jak Test Plan czy Test Raport.

Co ma wpływ na naszą pracę?

- **Organizacja, w której pracujesz** – możesz spotkać się z sytuacją, w której Twoja firma będzie miała zdefiniowane Test Policy, Test Strategy albo ogólny Proces testowy. Z tych dokumentów może wynikać, że przypadki testowe powinny być przygotowywane.

- **Klient, dla którego projekt jest realizowany** – sam klient też czasami wymaga dostarczania pewnej dokumentacji związanej z testami. Powody bywają różne: od braku zaufania do naszych testów, po konieczność dokumentowania całego procesu wytwarzania oprogramowania. Może być to podyktowane potrzebami jego klientów lub audytów związanych z certyfikacją. Czasami warto przypadki przygotować jako dowód tego, co zostało zrobione, w razie gdyby klient powiedział „sprawdzam".

  I tutaj mała gwiazdka. Czasami możesz spotkać się z taką sytuacją – dla klienta oznaczenie przypadku testowego na pass może nie być wystarczające. Pojawia się jeszcze pojęcie test evidence, które oznacza dostarczenie dodatkowej informacji, np. zrzutu ekranu, logów lub innego dowodu wykonania testu. Najczęściej taki dowód powinien przedstawiać rezultat oczekiwany, ale może też być potrzebny bardziej szczegółowy z uwiecznieniem wyniku dla każdego kroku.

- **Czas trwania projektu** – gdy masz do czynienia z krótkim projektem, to może nie być czasu ani sensu inwestować w przypadki testowe. Wtedy lepiej skupić się na testach eksploracyjnych niż tracić cenne godziny na dokumentowanie każdego kroku. Dla dłuższych projektów warto zdefiniować sobie chociażby regresję lub inne powtarzalne testy. Przygotowane przypadki mogą też być bazą do ich automatyzacji.

- **Wiek testowanego systemu** – możesz mieć okazję (tak jak ja kiedyś) pracować ze starym systemem tzw. legacy, gdzie jedyną dostępną dokumentacją jest kod aplikacji, a wiedza na temat poszczególnych funkcjonalności bywa mocno ograniczona. W takim wypadku zdobycie informacji na temat tego, jak coś działa i jak to przetestować, jest bardzo czasochłonne. Należy pilnować, by wysiłek nie poszedł na

marne. Trzeba więc udokumentować jego rezultaty, np. w postaci przypadków testowych, wiki lub testów automatycznych.

- **Metodyka, w której projekt jest realizowany** – jeśli masz do czynienia z pracą w sprintach, które kończą się sukcesem, to na początku kolejnego powinieneś mieć wolną chwilę, zanim jeszcze dostaniesz coś nowego do testów. Ten czas można wykorzystać na przygotowanie test casów lub napisanie testów automatycznych.

- **Twoje doświadczenie** – sam dobrze wiesz lub po pewnym czasie będziesz wiedział, czego potrzebujesz, by Twoja praca była efektywna. Przypadki testowe mogą pomóc dostarczyć zainteresowanym osobom potrzebnych informacji na temat testowanej aplikacji. Na ich bazie można przygotować metryki pokazujące np. liczbę planowanych i wykonanych test casów, liczbę tych, które zakończyły się sukcesem i tych, które nie przeszły. Takie metryki mogą być przygotowane dla całej wersji aplikacji, poszczególnych funkcjonalności lub wymagania.

Czy zatem warto pisać przypadki testowe?

Podsumowując, to czy i jaką dokumentację będziesz przygotowywać, może być narzucone lub zależeć od Ciebie, ale bardzo silnie zależy od kontekstu. Jest to zdecydowanie dobra metoda na spisanie zdobytej wiedzy na temat testowanej aplikacji. Może też pomóc w rozpoczęciu automatyzacji, bo będziesz mieć już opisane scenariusze, które można później oskryptować.

W ten sposób udokumentujesz też testy, które wykonałeś. Możesz wykorzystać tę dokumentację jako dowód, gdy później coś się wysypie. Przypadki testowe mogą się też przydać w procesie wprowadzania nowych członków zespołu do projektu lub gdy będziesz potrzebować chwilowego wsparcia, np. podczas regresji. Jeśli nikt nie narzuca konkretnej formy dokumentacji, warto rozważyć mapy myśli, które zdecydowanie się sprawdzają.

Czasami jednak możesz mieć okazję realizować projekt dla klienta, który nie wymaga dostarczenia jakiejkolwiek dokumentacji. Podobnie może być w przypadku firm produktowych, w których sposób pracy zależy w dużej mierze od upodobań zespołu. Wtedy warto rozważyć, czy inwestować czas w pisanie test casów, czy raczej skupić się na testowaniu. Jak pisze Michael Bolton na swoim blogu, test case to nie testowanie.

Zgłoszenie defektu

Po przeczytaniu kilku poprzednich stron wiesz już, jak powinien wyglądać przypadek testowy. Kolejnym etapem w pracy testera może być wykonanie testu i zaraportowanie błędów w momencie zaobserwowania anomalii. Jak powinno wyglądać takie zgłoszenie i jakie elementy zawierać? O tym za chwilę. Zajrzyjmy ponownie do „Słownika wyrażeń związanych z testowaniem" SJSI i sprawdźmy definicję defektu.

Defekt to wada modułu lub systemu, która może spowodować, że moduł lub system nie wykona zakładanej czynności.

Definicja definicją, a życie życiem. Często możesz spotkać się z pojęciami takimi jak defekt oraz błąd (bug), które chociażby według ISTQB to osobne zagadnienia. W życiu codziennym jednak te pojęcia są używane zamiennie i oznaczają nieprawidłowe zachowania systemu. Zwykle możemy mówić o błędzie w działaniu aplikacji, gdy testowana funkcjonalność zachowuje się niezgodnie ze zdefiniowanymi wymaganiami.

Poniżej znajdziesz przykład zgłoszenia defektu. Najpierw jeszcze kilka słów o tym, dlaczego tak ważne jest, aby dbać o jakość zgłoszenia. Jest to istotne ze względu na zespołowy charakter pracy. Takie zgłoszenie, zanim zostanie zamknięte, może przejść przez kilka osób. Czasami może się zdarzyć tak, że są to osoby, które nie znają tej części aplikacji, której błąd dotyczy. Warto więc zadbać o to, by opis był zrozumiały dla każdego. Uwierz mi, Tobie też to pomoże w momencie, kiedy błąd trafi do Ciebie do retestów po kilku tygodniach lub miesiącach od zgłoszenia. Warto wiedzieć, że niektóre zgłoszenia mogą być przeglądane przez klienta w trakcie spotkania Defect Triage. Wtedy także dokładny i zrozumiały opis jest bardzo przydatny.

Poniżej znajdziesz przykład zgłoszenia błędu. Przytoczony defekt został wykryty podczas wykonania przypadku testowego podanego w rozdziale wcześniej. Błąd dotyczy zadania #20 w aplikacji Mr Buggy 3.

Summary	Można wykonać przelew z ujemną wartością pola *Kwota*		
Description	**Kroki reprodukcji:** 1. Wprowadź poprawny numer konta w pole *Nr rachunku odbiorcy* np. 83109069928729691303181767 2. Wypełnij pole *Nazwa odbiorcy* dowolną wartością 3. Wypełnij pole *Tytuł przelewu* dowolną wartością 4. Wprowadź ujemną wartość w pole *Kwota* np. -5 5. Wciśnij przycisk *OK* 6. Wciśnij przycisk *Potwierdź* **Rezultat:** Pojawia się informacja *Przelew został przekazany do realizacji.* Wartość pola *Dostępne środki* została zwiększona o wprowadzoną kwotę przelewu **Rezultat oczekiwany:** Brak możliwości wprowadzenia ujemnej kwoty lub komunikat walidacyjny po wciśnięciu przycisku *OK*		
Configuration	Windows 10		
Priority	Krytyczny	**Reporter**	Krzysztof
Severity	Wysoki	**Status**	Open
Version	1.0.0.108	**Attachments**	przelew_ujemna_kwota.png

Najważniejsze informacje, które powinno zawierać poprawne zgłoszenie błędu, wymieniłem powyżej. Poza tym przydadzą się też różne załączniki: logi, screeny albo nawet filmiki. Te ostatnie dopiero niedawno zacząłem bardzo doceniać. W jednym z projektów testowałem rozwiązanie dostarczane przez zewnętrzną firmę. Była to dla mnie nowość przede wszystkim dlatego, że nie miałem możliwości bezpośredniego kontaktu z programistami. Komunikacja była więc bardzo ograniczona, przez co jakość zgłoszeń była niezwykle ważna. Mówi się, że jeden obraz wart jest więcej niż tysiąc słów. Filmiki rzeczywiście ułatwiały zgłoszenie defektów zwłaszcza przy bardziej skomplikowanych ścieżkach i czasami trudnym do opisania zachowaniu aplikacji.

Powyżej wymieniłem najbardziej powszechne elementy zgłoszenia. W praktyce często dodaje się jeszcze inne istotne informacje:

- faza testów, w której błąd został wykryty, np. UAT, SIT, development
- czy błąd został znaleziony przez testy automatyczne czy manualne
- z jakim typem wymagań błąd jest związany, np. funkcjonalne, bezpieczeństwa, wydajności.

Informacje te mogą pojawiać się w formie tagów lub labelek w zależności od użytego narzędzia. Posiadanie tych danych pozwala na bardziej zaawansowane analizy występowania błędów.

Severity vs Priority (Ważność vs Priorytet)

Jak pewnie zauważyłeś, w zgłoszeniu defektu pojawiają się takie atrybuty jak Priorytet oraz Ważność. Czas wyjaśnić, czym są te atrybuty i jaka jest między nimi różnica.

Priorytet błędu określa kolejność, według której błędy powinny być poprawiane. Niektóre defekty mogą blokować testowanie danej funkcjonalności. W takim przypadku, nadając najwyższy priorytet, możemy przekazać reszcie zespołu informację, że poprawka tego błędu powinna zostać dostarczona w pierwszej kolejności. Przykładowe priorytety zostały przedstawione na wykresie w podrozdziale dotyczącym metryk – Blocker, Critical, Major, Minor.

Ważność błędu określa, jak bardzo problem jest istotny z punktu widzenia biznesu. Jego wartość jest często powiązana ze stratami finansowymi lub wizerunkowymi. Przykładowe wartości to: Critical, Major, Minor.

Często krytyczny priorytet przekłada się na ważność na poziomie krytycznym, ale nie zawsze tak jest. Spójrz na przykłady poniżej.

Priorytet = krytyczny, Ważność = niska

Przykładem takiego błędu może być niewłaściwe logo na stronie głównej testowanej aplikacji. Z jednej strony nie powoduje to problemu w używaniu aplikacji, ale może wpłynąć na wizerunek firmy.

Priorytet = niski, Ważność = wysoka

Załóżmy, że testujesz aplikację, w której nie działa funkcjonalność wykorzystywana pod koniec roku kalendarzowego. Przeprowadzasz testy w połowie roku. Fakt, że nie działa ważna część systemu, powoduje, że ważność jest wysoka, ale ze względu na dużą ilość czasu defekt może zostać poprawiony w kolejnej wersji.

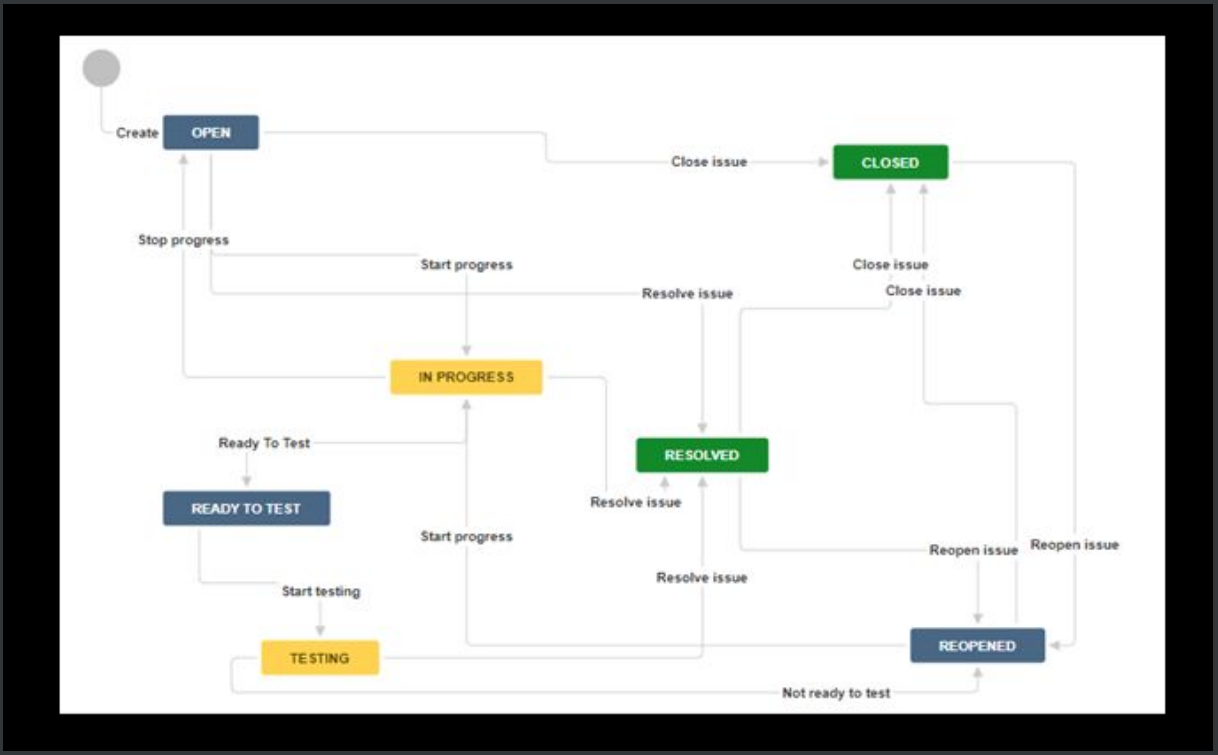
Lista używanych wartości dla obu parametrów może się różnić w przypadku różnych organizacji. Może też zależeć od użytego narzędzia, które ma zdefiniowane pewne domyślne ustawienia. Często wartości te są konfigurowalne.

Zapamiętaj powyższe przykłady oraz poszukaj więcej w internecie. To zagadnienie dość często pojawia się na rozmowach rekrutacyjnych.

Cykl życia defektu

Poniżej znajdziesz diagram z Jiry przedstawiający przykładowy cykl życia defektu. Dlaczego przykładowy? Tak jak już wspominałem na kartach tej książki, nie ma jednej sprawdzonej odpowiedzi na to pytanie. Cykl życia może się znacząco różnić między organizacjami i zespołami. Z pewnością po wpisaniu takiego hasła w wyszukiwarce pojawi się wiele różnych diagramów. Być może każdy z nich sprawdza się lepiej w innej sytuacji. Nie ma to jednak większego znaczenia dla dalszych rozważań. Warto wiedzieć, że tak jest. Tak samo jak to, że błędy mogą być zgłaszane przez dowolnego członka zespołu, a nie tylko przez

testera. Zresztą będziesz mieć też do czynienia ze zgłoszeniami od klienta. Te są zwykle bardzo ciekawe, ale nie będę zdradzał tajemnicy. Po prostu trzeba samemu przez to przejść.



Wróćmy jednak do diagramu i przeanalizujmy, co się na nim dzieje.

- Zgłoszony defekt na początku swego życia ma status **Open**.
- W kolejnym kroku może on zostać zamknięty (**Closed**) albo programista zacznie nad nim pracować (**In progress**). Jeśli przyjrzysz się dokładniej, zauważysz, że błąd może zostać zamknięty z poziomu wielu statusów. Powody mogą być różne, np.: zgłoszenie jest duplikatem; po dyskusji z analitykiem lub klientem okazuje się, że to nie jest błąd; defekt jest stary i został poprawiony przy okazji innych zmian; nie da się zreprodukować defektu.
- Programista zmienia status na **Ready To Test** w momencie, gdy błąd został poprawiony i można przystąpić do retestów.
- Status **Testing** oznacza, że błąd jest w trakcie weryfikowania.
- Jeśli w trakcie retestów okazuje się, że błąd nadal występuje, to tester oznacza go jako **Reopened** i cała zabawa zaczyna się od nowa.
- Jeśli jednak błąd został poprawiony, tester zmienia jego status na **Resolved**. W tym statusie błędy zwykle czekają na wdrożenie danej wersji z poprawkami i dopiero wtedy status zmieniany jest na **Closed.**

KRZYSZTOF JADCZYK – ekspert w dziedzinie testowania oprogramowania z kilkunastoletnim doświadczeniem. Brał udział w kilkudziesięciu projektach realizowanych w różnych technologiach (web, mobile, desktop, Salesforce, Mendix). Jako lider budował zespoły testowe oraz rekrutował testerów. Twórca bloga bugfreeblog.com, który powstał z pasji testowania. W książce dzieli się swoim doświadczeniem, dzięki czemu pomaga wszystkim zainteresowanym stawiać pierwsze kroki w zawodzie testera.

Więcej na

ISBN 978-83-955930-0-0
9 788395 593000