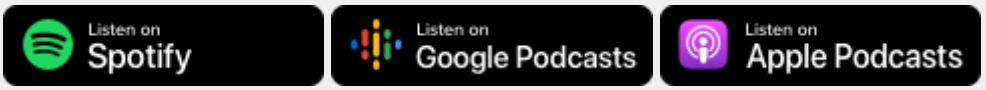
Pierwsze kroki w IT

PODCAST



Na czym polega praca testera oprogramowania?

Gość: Maciej Kusz

Wszystkie polecane materiały i linki znajdziesz na stronie odcinka:

https://devmentor.pl/b/na-czym-polega-praca-testera-oprogramowania

Dziś moim gościem jest Maciej Kusz. Maciej opowie nam o zawodzie testera oprogramowania. Macieju, dziękuję, że przyjąłeś moje zaproszenie na rozmowę.

Dzięki za zaproszenie. Miło mi i miejmy nadzieję, że rozmowa się uda.

Myślę, że na pewno się uda. To może zaczniemy od Twojej osoby. Powiedz nam, co łączy Cię z branżą IT?

To jest zawsze pytanie, na które można długo odpowiadać, ale tak, w dużym skrócie. Od 15 lat pracuję jako tester oprogramowania. W zasadzie zaraz po studiach ta przygoda się rozpoczęła. Od ponad 10 lat automatyzuję testy z użyciem Pythona. Po godzinach jestem twórcą bloga testerembyc.pl. Poza tym, jeszcze dodatkowe atrakcje typu współorganizator meetupu Ślonzacz-QA. Ostatnio też trzeci raz z rzędu jestem współorganizatorem konferencji ConSelenium. A dodatkowo, jeszcze jak chcę sobie trochę poprogramować, to tworzę taki projekt open source, który jest zestawem pluginów do MkDocs, czyli takiego narzędzia do tworzenia dokumentacji. Nazywa się to MkDocs Publisher, więc jeżeli ktoś chce, to zapraszam.

Od razu podpytam. Kiedy najbliższe meetupy? Może ktoś by chciał się wybrać.

Generalnie większość meetupów w okresie wakacyjnym zawiesza działalność, bo urlopy, więc raczej jest mało osób. Większość meetupów gdzieś tam we wrześniu wraca i my też planujemy wrócić we wrześniu. Nie wiemy dokładnie jeszcze kiedy. Mamy już wstępnie prelegentów i firmy, w których być może nam się uda spotkać. Nie mamy konkretnych dat, ale we wrześniu na pewno coś się będzie pojawiało, więc możecie śledzić.

To może chociaż miejsce?

Na pewno Katowice, bo Ślonzacz-QA jest w Katowicach. Która firma? Nie wiem. Mamy kilka do wyboru. Być może będzie to siedziba C, bo taki był wstępnie plan. A do potwierdzenia, że tak powiem, na socialach Ślonzacz-QA, bo tam wszystko widać.

Tak jest. To śledźcie od razu, zapiszcie sobie w kalendarzach, żeby sprawdzić na początku września.

A ja zacznę od pytania wyjaśniającego, bo wśród stanowisk testerskich znajdziemy takie jak tester manualny, automatyzujący, no i również QA. Wyjaśnisz nam, czym zajmują się takie osoby na tych stanowiskach i jakie są różnice?

Jasne. Ja zawsze staram się to tłumaczyć trochę od końca, bo podałeś rolę QA na końcu, a to jest rola, z którą zawsze najwięcej osób ma problem. QA jest bardzo szeroko rozumiane, a to nie jest do końca tak, jak być powinno.

QA to jest skrót od Quality Assurance, czyli to jest rola związana z zapewnieniem jakości produktu od strony procesu. To nie jest konkretnie osoba, która sprawdza, czy dany produkt faktycznie działa. To jest osoba, która ma za zadanie zbudować cały proces związany z jakością, czyli czy mamy dokumentację odpowiednio przygotowaną, czy po procesie przygotowania paczki software'u mamy odpowiednie testy, manualne, czy automatyczne, czy ileś poziomów. To różnie bywa. I taka osoba definiuje pewne dodatkowe rzeczy. Na przykład, czy te testy przeszły, czy mają odpowiedni poziom, czy pewne rzeczy są sprawdzone. Ona się nie zajmuje stricte testowaniem, ale całą otoczką dookoła testowania.

Czy to jest osoba, która wcześniej była testerem takim manualnym lub automatyzującym?

W większości wypadków tak jest, że Quality Assurance wychodzi bardzo często naturalnie z poziomu zwykłego testera. Jeżeli ktoś zajmuje się

testowaniem aplikacji, to zaczyna rozumieć pewne rzeczy, jak one działają, co na co ma wpływ, na co trzeba zwrócić uwagę. I jeżeli ktoś lubi takie rzeczy, to zaczyna naturalnie przechodzić z roli zwykłego testera właśnie w rolę QA. Są też osoby, które oczywiście potrafią od razu wskoczyć w tę rolę, bo bardziej to czują, ale najbardziej naturalny proces to jest ewolucja, a nie od razu wskakiwanie na jakiś tam poziom.

Kolejne dwie role, o które pytałeś, czyli tester manualny i tester automatyzujący, to są role, gdzie jedna powinna wynikać z drugiej. Jeżeli zaczynamy testować aplikację, sprawdzać, czy ona jest poprawnie napisana, czy kliknięcie w dany przycisk powoduje to, co ma zrobić. Na przykład dodanie produktu do koszyka albo zamówienie jakiegoś produktu czy cokolwiek innego. To zaczyna się od testera manualnego. Tester manualny wykonuje te akcje.

To, czym on się jeszcze zajmuje, to tworzenie scenariuszy testowych. Samo testowanie, klikanie po aplikacji i tak dalej, to jest proces bardzo powtarzalny, ale żeby to zrobić dobrze, to musimy wiedzieć, co zrobić. Musimy sobie przygotować jakiś scenariusz testowy. Taki scenariusz przygotowuje się najczęściej, tak by było najlepiej, jeżeli mamy specyfikację produktu, czyli opis, co ten produkt w danym miejscu powinien robić. Jeżeli mamy taką specyfikację, mniej lub bardziej formalną, to jesteśmy w stanie jako tester manualny przygotować scenariusz, dobrze go opisać, wszystkie kroki, które musimy przeprowadzić w aplikacji, żeby sprawdzić, czy dana rzecz działa tak, jak ma działać. Później przechodzimy do kolejnego kroku, czyli testowania, wykonania tego scenariusza.

Kolejny krok, dodatkowa rzecz, która jest najbardziej żmudnym procesem w życiu testera manualnego, czyli testowanie po raz kolejny tego samego. Jeżeli wydajemy aplikację w wersji pierwszej, mamy ją przetestowaną, przekazujemy ją do klienta, wszystko działa, ale klient zgłosi nam jakieś błędy albo jest jakiś rozwój aplikacji i dodajemy nową funkcjonalność, to tester przy wersji drugiej musi znowu sprawdzić, czy te rzeczy z wersji pierwszej dalej działają, czy nie wprowadzona została tak zwana regresja. Tester musi przejść te wszystkie wcześniejsze scenariusze, jeżeli jest nowa funkcjonalność, dołożyć nowe scenariusze. Jest to proces bardzo powtarzalny. Tu dochodzimy do kolejnej roli, czyli do roli testera automatyzującego.

To może bym tylko wszedł w słowo, zanim zaczniesz opowiadać, bo często też się słyszy o nazwie tester automatyczny. Czy to błąd w tłumaczeniu, czy to gdzieś tam ewoluowało? Jak to rozumieć?

To bardzo potoczna nazwa. I tutaj chyba trzy czy cztery lata temu duża część środowiska testerskiego, osób, które troszeczkę więcej rozumiały, bardzo walczyła z tym pojęciem. To nie jest tester automatyczny, bo my nie jesteśmy automatem, tylko my automatyzujemy testy. Więc poprawna nazwa to tester automatyzujący, nie tester automatyczny.

Dochodzimy do tej roli. Tester manualny robił różne rzeczy w sposób powtarzalny. No i teraz fajnie by było sobie troszeczkę ułatwić pracę, czyli te rzeczy, które powtarzamy przy każdym release'ie oprogramowania. Fajnie by było je zautomatyzować, żeby nie trzeba było tego robić ręcznie w żmudny, powtarzalny sposób. Tester automatyzujący w dużym skrócie automatyzuje scenariusze testów manualnych. To jest najprostsza definicja testera automatyzującego.

Trzeba mieć tutaj na uwadze jedną rzecz: tester automatyzujący nie wykrywa tak wielu błędów albo tych samych błędów, co tester manualny. Dlaczego? Dlatego, że test automatyczny to test, który powinien być pisany dla funkcjonalności, która jest już w jakiś sposób stała, czyli została już sprawdzona przez testera manualnego. Jest już, powiedzmy, gotowa dla końcowego klienta. Wiemy, że nie będzie się już za bardzo zmieniała. Taki test automatyczny danej funkcjonalności, danego scenariusza testowego, upewnia się w sposób powtarzalny, że to, co zostało już napisane, nie zostało zdegradowane. Czyli czy nie mamy znowu tej regresji, czy nie wprowadzamy błędów w istniejącej funkcjonalności.

Trzeba mieć na uwadze, że test automatyczny to check. To nie jest typowy test, który wykrywa nowe błędy. On raczej wykrywa, czy nie zostały wprowadzone stare błędy albo nawet nowe błędy, ale w funkcjonalności, która już powinna działać.

I teraz pojawia się pytanie: Czy dobry tester automatyzujący musi być wcześniej testerem manualnym? Moim zdaniem tak. Bo jeżeli chcemy coś dobrze automatyzować, to musimy też wiedzieć, jak to przeprowadzić w sposób manualny. Nawet jeżeli mamy scenariusz dobrze rozpisany, mogą być sytuacje, że test automatyczny w pewien sposób odbiega w pewnych aspektach od testu manualnego, ale nie możemy pewnych elementów pominąć, tylko musimy je zrobić w trochę inny sposób. Bo nie wszystko da się wprost zautomatyzować.

Często też w mniejszych firmach nie ma typowej roli testera manualnego, tylko ktoś zatrudniany na stanowisko testera jest od razu zatrudniany do roli testera automatyzującego. Wtedy taka osoba, poza

pisaniem tych testów automatycznych, musi też umieć sobie przygotować cały scenariusz, który normalnie przygotowuje tester manualny. Umiejętność testowania manualnego wśród testerów automatyzujących powinna być na dobrym poziomie.

Tutaj jeszcze jest kolejny aspekt, bo tester automatyzujący to bardzo szerokie pojęcie. Potocznie przyjmuje się, że tester automatyzujący to osoba, która testuje front-end aplikacji webowej. To nie jest do końca tak, bo mamy jeszcze często REST API, w testach automatycznych mamy też takie rzeczy jak testowanie wydajności. Pod pojęciem testera automatyzującego można bardzo dużo rzeczy jeszcze ukryć.

To wejdę w słowo, bo już zacząłeś ten temat, ale chciałbym się zapytać o rodzaje testów, które są przeprowadzane. Jakbyś mógł je nazwać i opisać, żeby osoby, które się interesują albo które wchodzą w tę branżę, wiedziały mniej więcej, o co chodzi. Wiadomo, że nie będziemy się teraz doktoryzować na ten temat.

Rodzajów testów jest bardzo dużo, bo można o tym napisać książkę i takie książki już istnieją. Nie pamiętam tytułu, ale Adam Roman ma książkę, która dużo takich rzeczy opisuje.

Pierwszym typem, z którym spotykamy się, gdy zaczynamy pracę jako tester manualny, bo taka powinna być według mnie naturalna ścieżka, o czym mówiliśmy wcześniej, to są testy eksploracyjne. Dostajemy aplikację, specyfikację, która nam opisuje i zaczynamy używać tej aplikacji. Eksplorujemy, sprawdzamy, co możemy zrobić, czy to, co jest opisane w specyfikacji, faktycznie działa. Sprawdzamy, jaką akcję możemy wykonać w trochę inny sposób, jaki ona efekt przyniesie, sprawdzamy co możemy zrobić, a czego nie, jaki jest tego wynik itd. To

jest naturalny proces. My nawet jako zwykli użytkownicy, którzy nie są testerami, bardzo często mamy takie sytuacje. Dostajemy nowy telefon, zaczynamy go używać, to eksplorujemy, sprawdzamy.

Od razu przyszło mi do głowy, że obecnie producenci gier tak traktują swoich klientów.

Testowanie na produkcji, tak?

Tak, tak.

Dokładnie tak to działa. Następnym elementem jeżeli już przeprowadzimy testy eksploracyjne i stworzymy scenariusze do tych testów, to mamy etap powtarzania, czyli sprawdzamy, czy nie wprowadziliśmy nowych błędów. To nazywa się testami regresji. To pojęcie już wcześniej pojawiło się w naszej rozmowie. Cyklicznie sprawdzamy, czy dana funkcjonalność dalej działa, czy nie zostały wprowadzone nowe błędy, czy coś się nie zmieniło.

Tutaj zrobię małą dygresję. Po to tworzy się testy automatyczne, żeby odzyskać czas testera z testów regresji. Skoro coś robimy powtarzalnie, możemy to zautomatyzować, więc czas, który tester wcześniej poświęcał na regresję, możemy przeznaczyć na inne testy, chociażby na eksplorację nowych obszarów, nowych funkcjonalności.

Pojęciem, które przy okazji testów regresji i testów automatycznych się pojawia, są tzw. smoke testy. Smoke testy to podzbiór testów, które sprawdzają najbardziej kluczową funkcjonalność w aplikacji. Na przykład, jeżeli mamy sklep internetowy, najbardziej kluczową funkcjonalnością z punktu widzenia sklepu internetowego jest proces od

dodania produktu do koszyka poprzez zamówienie tego koszyka i opłata oraz sprawdzenie czy płatność przeszła.

Proces dostarczenia produktu to jest już coś, co jest przeważnie wydelegowane na zewnątrz.

Najważniejsze, żeby kasa się zgadzała.

Dokładnie. Z punktu widzenia twórcy, czy tam właściciela sklepu, ten proces zamówienia i wpłynięcia pieniędzy na konto to jest najbardziej kluczowy proces. Więc jednym z takich smoke testów będzie właśnie taki proces zamówienia.

Czyli o co chodzi w ogóle? Po co są smoke testy? Smoke testy bardzo często są na etapie w którym bardzo często wydajemy nowe wersje oprogramowania i chcemy się upewnić. Powiedzmy, że nawet aplikacja została wgrana na tak zwaną produkcję, czyli została udostępniona klientowi, to smoke testy często są w taki sposób przygotowywane, że możemy je uruchomić nawet na produkcji i upewnić się, że najbardziej kluczowe funkcjonalności po prostu działają. Często się to robi w ten sposób, że jest podzbiór wybranych testów. Na przykład mamy 300 testów automatycznych, ale 30 testów to są te nasze smoke testy.

To, jak o tym opowiadałeś, to trochę mi się skojarzyło z antywirusem, że ma pełne skanowanie albo tylko takie najważniejsze.

To dokładnie na tej samej zasadzie działa. No, tutaj CrowdStrike ewidentnie nie miał czegoś takiego, no i był problem, tak?

Tak jest.

Kolejny element układanki to testy integracyjne. Testy integracyjne to testy, które sprawdzają, czy wszystkie elementy oprogramowania, które ktoś pisze ze sobą działają tak, jak powinny działać. O co tutaj chodzi? W większości aplikacji teraz, powiedzmy skupiając się tylko na aplikacjach webowych, mamy aplikację webową podzieloną na kilka komponentów, czyli mamy tak zwany frontend, czyli to, co widzi użytkownik w przeglądarce. Mamy tak zwany backend, gdzie się wykonują wszystkie akcje biznesowe, zapisują się dane o transakcjach, o opłatach i tak dalej. Frontend z backendem muszą ze sobą w dobry sposób rozmawiać, żeby działania które użytkownik zrobi w przeglądarce, na serwerze zdalnym wykonały się w odpowiedni sposób.

Jednym z takich działań, żeby sprawdzić, czy te wszystkie elementy, które wchodzą w skład tego oprogramowania, dobrze ze sobą działają, są właśnie testy integracyjne. Sprawdzamy, czy wszystko się dobrze ze sobą dogaduje i efekt końcowy jest taki, jaki chcemy. Tutaj możemy znowu wrócić troszeczkę do tych wcześniejszych smoke testów, które mogą wynikać z testów integracyjnych. Smoke test powinien sprawdzić coś po integracji, czyli całą najbardziej kluczową ścieżkę biznesową.

Kolejny etap. Tutaj coś, co teraz jest bardzo na topie, jest bardzo dużo ofert pracy. Kilka lat temu testerzy automatyzujący byli na topie pod kątem zatrudnienia i stawek, tak teraz właśnie są testerzy bezpieczeństwa (security). Te testy są dosyć specyficzne, bo po pierwsze trzeba wiedzieć troszeczkę więcej niż taki tester zwykły, manualny czy automatyzujący. Musimy się zagłębić w to, jak aplikacja jest zbudowana, gdzie są ewentualnie tak zwane backdoory, o których wszyscy mówią. Czyli gdzie mamy coś dotknąć w aplikacji, żeby zrobić

coś więcej niż aplikacja przewidziała. Na przykład mieć dostęp do bazy danych wszystkich użytkowników albo tego typu rzeczy. Czy możemy bezpośrednio rozmawiać z jakimiś komponentami systemu, które nie powinny być widziane przez klienta.

Tu jest cała oddzielna w ogóle dziedzina wiedzy z zakresu testowania. Testy bezpieczeństwa bardzo często są też testami takimi typowo manualnymi, bo jest je ciężko zautomatyzować, ale można pisać testy automatyczne do security, jeżeli już mamy dobrze rozpisany scenariusz, wiemy jak pewne rzeczy dotknąć, znamy przykłady problemów z bezpieczeństwem w danej aplikacji, więc można pewne testy pod kątem security również zautomatyzować.

Kolejny rodzaj testów to coś, co teraz jest może nie aż tak na topie, ale powraca w tak zwany czarny piątek. Testy wydajności. O co tutaj chodzi? Generalnie chodzi o to, żeby sprawdzić, czy nasza aplikacja, jest przygotowana do obsłużenia większej ilości użytkowników. Na przykład w ten czarny piątek, gdzie wszyscy w sklepach rzucają się, zamawiają, bo są promocje. Musimy sprawdzić, czy dana aplikacja, która normalnie obsługuje, powiedzmy, 100 użytkowników w ciągu godziny, jest w stanie obsłużyć 10 tysięcy użytkowników w ciągu godziny.

Testy wydajności to w większości wypadków, powiem szczerze, chyba we wszystkich przypadkach, bo nie kojarzę ze swojej kariery, żeby ktoś robił to w sposób manualny, są to testy automatyczne, które są automatyzowane przy użyciu kilku dostępnych narzędzi na rynku. Zresztą chyba w późniejszej części gdzieś tam o tym jeszcze powiemy, jak będziemy rozmawiać o narzędziach, więc wrócimy do tego tematu.

Czy obecnie ten temat nie jest dość łatwy do rozwiązania? W sensie, że mamy chmurę i to po prostu, w cudzysłowie, samo tam się replikuje?

I tak i nie. Pod kątem skalowania, tak zwanego skalowania aplikacji, czyli przystosowania aplikacji do większego obciążenia, to teoretycznie jest proste, w praktyce niekoniecznie. Tutaj muszą mechanizmy wykrywania większego obciążenia w odpowiedni sposób działać, aplikacja musi być w odpowiedni sposób napisana, żeby dało się ją wyskalować do większego ruchu, więc tutaj musi się kilka elementów zadziać.

Z punktu widzenia samych testów, są narzędzia, które potrafią w sposób automatyczny przygotować takie testy, ale wydajność jest w ogóle, tak jak security, oddzielną poddziedziną testowania i tutaj trzeba dosyć sporo wiedzieć. Samo obciążenie aplikacji w pojedynczych miejscach, czyli powiedzmy mamy ten proces, o którym mówiliśmy, zamawiania jakiegoś produktu, dodanie czegoś do koszyka, potwierdzenie, że koszyk jest okej, zamówienie produktów i płatność, to jest coś, co jest podzielone na pewne etapy. Z punktu widzenia aplikacji, to nie jest tak, że my wysyłamy jeden, tak zwany, request w jedno miejsce i to się dzieje. Podczas tego procesu, gdzie użytkownik komunikuje się z frontendem i z backendem, jest wysyłanych kilka requestów. Jeżeli z punktu widzenia wydajności my tylko zapytamy jeden z tych elementów, czy on jest w stanie obsłużyć te 10 tys. użytkowników, to my nie mamy pełnego obrazu całego tego procesu, bo sprawdziliśmy tylko jedno miejsce, a nie 5 czy 10 miejsc na raz.

Stworzenie dobrych testów wydajności jest może nie trudne, ale jest dosyć skomplikowanym procesem, bo musimy wiedzieć, jak to przygotować, żeby obciążyć aplikację w sposób taki, żeby odzwierciedlić ten faktyczny ruch 10 tys. użytkowników. Trzeba umieć to zrobić.

Możemy jeszcze przejść do kolejnego etapu. Są pewne testy, które nie są często spotykane, ale są dosyć istotne. Na przykład testy certyfikacji oprogramowania. Niektóre urządzenia muszą mieć certyfikowany firmware po to, żeby w ogóle ten firmware mógł wyjść do klienta. Są urządzenia automatyki przemysłowej, które podczas procesu produkcji jakichś produktów mają styczność z ludźmi i te urządzenia muszą być certyfikowane pod kątem bezpieczeństwa. Zapewnienie jakości danego produktu musi być potwierdzone certyfikatem, bo to urządzenie może na przykład, źle zadziałać i obrócić ramię robota w niekontrolowany sposób i kogoś uderzyć albo zabić.

Są pewne dziedziny tworzenia oprogramowania czy testowania, w których muszą się wydarzyć testy certyfikacji po to, aby potwierdzić, że dany produkt czy dane oprogramowanie jest zgodne z jakimiś wymaganiami.

Oczywiście te testy, które tutaj wymieniliśmy, tych siedem czy osiem, to nie jest wszystko. Tych rodzajów testów jest dużo, dużo więcej. Te, o których myśmy tutaj powiedzieli, to są testy powiedzmy najczęściej spotykane. Tak jak mówię, w zależności jeszcze od firmy, od produktu, od wymagań zewnętrznych, może tych testów być dużo więcej i proces testowania to nie jest tylko testowanie w przeglądarce i klikanie, tylko ten proces może być bardzo rozległy. Może obejmować wiele aspektów czy rodzajów testowania naraz.

Powiedzieliśmy o stanowiskach, o testach, to może czas powiedzieć o zespole. w którym pracuje tester oprogramowania. Czy są to sami testerzy, czy może taka osoba jest częścią zespołu developerskiego? Jak to wygląda najczęściej?

Można zacząć tę odpowiedź od słynnego: *to zależy*. Zależy od tego, w jakiej jesteśmy firmie, co testujemy, jaki jest produkt, jaki jest proces. Tego: *to zależy* jest tutaj dużo, ale powiedzmy, że najczęściej jest tak, że tester jest częścią zespołu. Takie jest założenie, mamy jakiś zespół, w którym jest, powiedzmy 3-4 developerów i na przykład dla tych developerów jest dedykowany jeden tester, który przeprowadza testy tego, co Ci developerzy napiszą.

Ale są też sytuacje, w których zespół developerski jest oddzielony zupełnie od zespołu testerów i zespół testerów to jest na przykład cały dział, gdzie mamy 20 testerów, którzy są oddelegowani do testowania firmware'u jakichś urządzeń albo jakiejś bardzo rozbudowanej aplikacji. Często w przypadku testów wydajnościowych, o których mówiliśmy chwilkę wcześniej, też jest tak, że testerzy, którzy testują wydajność, są zupełnie oderwani od zespołu, bo żeby testować wydajność, po pierwsze aplikacja musi być już dobrze przetestowana przez testerów, którzy są częścią zespołu. To, co powiedziałem wcześniej, po pierwsze tworzenie testów wydajnościowych jest trudniejsze. Musimy mieć w miarę stabilną aplikację, czyli przetestowaną i dobrze działającą, po to, żeby nie marnować czasu na testowanie wydajnościowo czegoś, co nie działa. Więc tutaj może być taka sytuacja.

Podobnie może być z testami bezpieczeństwa. Testerzy, którzy testują bezpieczeństwo, są oderwani od całego zespołu takiego normalnego.

Teoretycznie najlepsza sytuacja, jeszcze wracając do tego, że tester jest częścią zespołu, jest taka, że w procesie tworzenia danej funkcjonalności, rozszerzania funkcji aplikacji, tester jest ostatnim elementem, który potwierdza, że dana funkcjonalność została poprawnie zaimplementowana. Takie są powiedzmy założenia tego tak zwanego zespołu Scrumowego. Ale są sytuacje, w których pomimo tego, że tester jest częścią takiego zespołu, to jego testy, są oderwane zupełnie od procesu tworzenia funkcjonalności. Na przykład testerzy manualni są częścią, ale testerzy automatyzujący już są troszeczkę odsunięci na bok, pomimo że są częścią zespołu. Ich testy są tworzone nie w tym samym sprincie, tylko w kolejnym. Z podobnych względów, jak w testach wydajnościowych, tworzenie testu automatycznego dla czegoś, co nie jest w miarę dobrze działające, jest stratą czasu i środków. Więc tu dużo zależy od tego, w którym miejscu, jakiego testera postawimy, to może być on częścią zespołu, ale nie musi.

To jak już wspomniałeś o sprincie, to może czas na zadanie pytania, jak się ma właśnie praca w Scrumie do pracy testera.

Mamy podejście typowo książkowe, tak jakby sobie tego Scrum życzył, czyli rola testera powinna być wymienna z rolą developera. Tester jest częścią zespołu, ale może też, czy powinien umieć programować. W rzeczywistości nie do końca to tak jest, bo poziom umiejętności programowania testerów, nawet automatyzujących, bardzo często jest zupełnie inny od developerów. I pomimo tego, że tester automatyzujący na przykład jest w stanie przeczytać kod, który stworzyli developerzy, niekoniecznie będzie w stanie go rozwijać, ale może tworzyć testy jednostkowe.

Tutaj jest coś, o czym nie mówiliśmy, czyli o testach jednostkowych. A nie mówiliśmy o nich dlatego, że często testy jednostkowe są jakby po stronie zespołu developerskiego, a nie testerów, ale testerzy też mogą je robić.

Kolejne podejście do roli testera to jest takie podejście, ja je nazywam prawie książkowe. To o czym już wspomniałem, tester jest częścią zespołu. I żeby powiedzieć, że dana funkcjonalność została już stworzona i dobrze działa, to tester musi przeprowadzić testy i powiedzieć, że jest OK. Ale nie ma tej wymienności z rolą developera. Czyli jest częścią zespołu, ale nie ma takiej typowo książkowej definicji wymienności z developerem.

No i jest jeszcze rzeczywistość, która stety, niestety w wielu wypadkach jest inna niż to, co mówiliśmy wcześniej. Podejście książkowe czy prawie książkowe, gdzie tester może być w dowolnym miejscu i w zależności od tego, co się dzieje i jak się dzieje, może zamykać potwierdzenie, że coś zostało dobrze napisane. Może być to, co powiedziałem wcześniej, przesunięty na przykład o jeden sprint, tak jak tester automatyzujący, który testy automatyczne dostarcza już po tym, jak funkcjonalność została przetestowana i potwierdzona, że działa. Albo jest testerem wydajności, który jest w ogóle oderwany od tego procesu i testy wydajności są po miesiącu. Bo proces tworzenia tych testów jest dużo bardziej skomplikowany. Więc znowu: *to zależy*.

Jak już wspomniałeś o testach jednostkowych i współpracy z programistami, to jak faktycznie wygląda ta współpraca. Gdzie jest granica między TDD, BDD i pracą testera?

To jest coś na co bardzo dużo testerów narzeka, na tę granicę. Zacznijmy od tego, że to też zależy. Zależy od dojrzałości zespołu. W idealnym świecie jest to, co powiedzieliśmy wcześniej, że tester jest częścią zespołu Scrumowego, czy w ogóle zespołu, który wytwarza oprogramowanie. Ta współpraca tutaj powinna bardzo płynnie przebiegać. Czyli jeżeli tester coś robi, to może przyjść do developera, zapytać się, pogadać i w ogóle.

Niestety czasem jest święta wojna pomiędzy testerami i developerami. Tester coś chce, a developer nie ma na to czasu albo nie do końca mu się chce albo uważa, że to jest w ogóle niepotrzebne. Są takie dziwne niesnaski, które nikomu nie służą i tutaj to, co powiedziałem, dojrzałość zespołu, im ktoś dłużej pracuje w branży i więcej rozumie, że testerzy grają do tej samej bramki co developerzy. Starają się pomóc, żeby była lepsza jakość produktu, żeby developer nie musiał trzy razy pracować, trzy razy tego implementować. Im szybciej developerzy to przetrawią i zrozumieją, tym ta współpraca na linii tester-developer lepiej się układa. To jest jeden z aspektów.

Kolejny aspekt, tutaj pytałeś o TDD. W ogóle czym jest TDD? TDD to jest Test-Driven Development, czyli tworzenie implementacji czegoś, do czego już mamy stworzony test. Tylko tutaj jest to, o czym wcześniej mówiliśmy, że te testy na poziomie TDD to nie są takie testy automatyzujące czy testy manualne, to są testy jednostkowe, które tworzy w większości developer, a nie tester.

Żeby zrobić dobre testy jednostkowe pod kod, który nie istnieje, musimy mieć czas na to po stronie developmentu, żeby znać całą architekturę aplikacji, wszystkie elementy, być dosyć dobrym programistą, żeby

wiedzieć, jak pewne rzeczy będą zaimplementowane. Żeby móc stworzyć najpierw dla nich testy jednostkowe. No i tu się pojawia pewien zgrzyt, bo w większości wypadków w projektach nie ma na to czasu, więc TDD bardzo często nie istnieje, bo raczej jest w drugą stronę, czyli najpierw tworzymy implementację, a później developerzy do istniejącej implementacji tworzą testy jednostkowe, czyli zupełne zaprzeczenie idei TDD.

Kolejny etap, o który tutaj pytałeś, to są BDD, czyli Behaviour-Driven Development. To troszeczkę wywodzi się z TDD, ale chodzi o to, że w procesie developmentu, czy w procesie tworzenia w ogóle całego oprogramowania, w większości wypadków nie bierze udziału tylko developer i tester, ale jeszcze musi być ktoś od strony biznesu, kto opisze daną funkcjonalność. BDD powstało po to, żeby ułatwić komunikację na linii biznes, developer, tester, żeby przy użyciu składni tak zwanego Gherkina, to jest taki syntax, ładnie się mówi, czyli składnia opisu User Stories. Używa się takich słówek jak: given, when, then. Mówimy na przykład: jeżeli mamy given użytkownika, który jest klientem sklepu i when, kiedy chce on zamówić jakiś produkt, then dodaje produkt do koszyka i coś tam się dalej dzieje.

Tworzymy taki opis ustrukturyzowany, który opisuje nam jak dana funkcjonalność ma działać. I teraz jeżeli mamy stworzone takie User Story przy użyciu tej składni Gherkina, to powinien tworzyć je biznes, developer na podstawie tego User Stories powinien stworzyć implementację kodu, a tester na podstawie tego powinien być w stanie stworzyć na przykład test automatyzujący. Tu się pojawia wiele różnych problemów na tej linii, dlatego ja nie jestem zwolennikiem w ogóle BDD i

dużo ludzi w świecie testów też nie. Po pierwsze, żeby BDD działało, to muszą być w tym procesie zaangażowane wszystkie trzy strony, czyli biznes, developer i tester.

Często niestety jest tak, że BDD powstaje od drugiej strony. Testerzy chcą wprowadzić BDD, bo ktoś im powiedział, że *BDD jest super i wprowadźcie BDD w testach automatyzujących*, no ale nie mamy tych pozostałych dwóch stron. Więc jaki jest sens wprowadzania czegoś, co jest po pierwsze dodatkową warstwą abstrakcji w kodzie, bo zaciemnia to, co implementujemy.

Po drugie, jakkolwiek nie patrząc, czy jest to od strony testera, czy jest to poprawna ścieżka, pojawiają się problemy z utrzymaniem spójności User Stories. O co tutaj chodzi? Jeżeli mamy te wszystkie: given, when, then. To one powinny być pisane w ten sam zunifikowany sposób. Jeżeli mamy użytkownika, którego opisujemy jako klienta sklepu, to ta fraza, ten given, który go opisuje, powinna być we wszystkich User Stories zrobiona tak samo.

Tu się pojawia problem, bo przychodzi nowy business owner albo ktokolwiek inny i zamiast tej samej składni zaczyna wprowadzać nową. Żeby to działało musimy zrobić albo nową implementację kroku albo musimy dodać, że ten kod, który mamy, który obsługiwał wcześniej taki given, to obsługuje jeszcze taki given. Zaczyna się to wszystko rozjeżdżać.

Co jeszcze mamy i jakie problemy? Ta generyczność właśnie tych kroków. Im większy jest projekt, tym bardziej się to rozjeżdża, więc koszt

utrzymania spójności w samych User Stories zaczyna przewyższać koszt tworzenia automatyzacji na przykład po stronie testów.

Są lepsze narzędzia. Jeżeli mamy proces, w którym nie bierze udziału business owner, developer i tester, mówię tutaj o samym BDD. BDD wychodzi od strony testów, to po co tworzyć coś takiego, skoro można bardzo podobny efekt od strony raportowania wyników testów uzyskać poprzez narzędzie które się nazywa Allure. Tam wystarczy wykorzystać taki dekorator, który się nazywa Step i opisywać dokładnie to samo, co jest w BDD. W efekcie końcowym raport, który pokazujemy czy to biznesowi, czy jakiemuś product ownerowi, czy komuś z szefostwa, kierownictwa, będzie wyglądał niemal identycznie jak raport wygenerowany przy użyciu BDD, ale zmniejszamy ten poziom skomplikowania kodu, bo wyeliminowujemy ten poziom abstrakcji na poziomie BDD.

Jest jeszcze jeden problem związany z samym BDD. Problem z przechowywaniem tych User Stories. Biznes je tworzy, to gdzieś musi to zapisać, zapisuje to w jakimś narzędziu, na przykład w Confluence, które jest bardzo popularne w branży IT.

Teraz tak. Mamy tamtą implementację tych kroków, czy opis tego User Stories, ale tester automatyzujący musi zaimplementować kod tych testów. Więc musi skopiować to User Stories do poziomu repozytorium z kodem testów, no i zaczyna się znowu ten problem synchronizacji tego wszystkiego. Gdzie jest źródło prawdy o tym, jak aplikacja ma działać? W Confluence wyedytowanie czegoś jest proste. Ale skoro biznes tam wyedytuje, to niekoniecznie zrobi to na poziomie kodu testów automatycznych. Można tutaj kombinować niby połączenie, że tylko w

kodzie jest implementacja bez User Stories, ale tu się znowu pojawia, że jak ktoś wstawi nowe słówko w kroku, który powinien być tak opisany, a jest troszeczkę inaczej, to przestaje to działać.

Problemów związanych z BDD jest więcej niż korzyści, przynajmniej moim zdaniem. Jeżeli proces jeszcze jest bardzo kulawy, czyli nie ma tych wszystkich trzech stron.

Jak już wspomniałeś o narzędziach, to jakie narzędzia są wykorzystywane w pracy testera?

Dużo zależy od samych testów, na jakim poziomie i kto testuje. Inaczej to będzie wyglądało na przykład dla testera manualnego, którego głównym narzędziem w przypadku testowania aplikacji będzie po prostu przeglądarka. Bo musi tę aplikację przeklikać i sprawdzić, jak to wygląda.

Bardzo często testerzy manualni, co tworzą? Tworzą między innymi scenariusze testów. Potrzebujemy jakiegoś narzędzia, które będzie służyło do przechowywania tych scenariuszy testów. Można użyć Worda, ale edycja Worda i jeszcze synchronizowanie tego między użytkownikami jest trudne. Są do tego specjalizowane narzędzia, na przykład TestRail, Xray, które potrafią to zrobić w sposób dosyć ustrukturyzowany, fajny. Można tam wersjonować, dodawać dodatkowe atrybuty, na przykład, że dany test został zautomatyzowany przy użyciu Playwright, Selenium czy czegokolwiek innego.

Oczywiście tester manualny potrzebuje jeszcze kolejnego narzędzia, czyli miejsca, gdzie zgłasza błędy. Najbardziej popularnym i najczęściej

spotykanym chyba jest Jira. Są też inne narzędzia, na przykład open source'owa Bugzilla, Redmine i kilka innych.

Narzędziem, które bardzo się przydaje testerowi manualnemu i automatyzującemu, do zgłaszania błędów jest narzędzie do tworzenia screenshotów. Jest cała masa tego typu narzędzi. Ja testuję na Macach i tam jest takie narzędzie, które się nazywa Shottr. Uważam, że to jest chyba jedno z najlepszych narzędzi jakie widziałem do tej pory na rynku, jeżeli chodzi właśnie o tworzenie screenshotów i dodawanie jakichś opisów na screenshot.

Z punktu widzenia testera automatyzującego będzie to troszeczkę inaczej wyglądało. Będą elementy wspólne, jak na przykład Jira czy TestRail, o których mówiliśmy w przypadku testera manualnego, czy to narzędzie do robienia screenshotów. Ale będziemy mieli jeszcze narzędzia trochę już z pogranicza programowania, czyli tego, co używa developer, jakieś IDE do tworzenia kodu. Będzie to Git do przechowywania tego kodu. Będą to jakieś narzędzia do CI/CD, na przykład Jenkins albo GitHub Actions.

Z punktu widzenia samych testów będą to bądź narzędzia, bądź biblioteki do przeprowadzania konkretnych testów na konkretnym poziomie. Jeżeli testujemy REST API, no to najbardziej popularnym narzędziem jest Postman, który ma swoje wady i zalety. Ostatnio bardzo dużo firm się zaczyna z niego wycofywać z tego powodu, że cały kod tworzony w Postmanie jest przechowywany w chmurze firmy, która tworzy Postmana, tutaj jest problem od strony ewentualnego security i tak dalej, więc część firm się z tego wycofuje.

Z punktu widzenia pisania kodu na poziomie testowania REST API, to w Pythonie jest biblioteka Request, w Javie jest taka biblioteka, która się nazywa REST Assured. W innych językach nie pamiętam, ale na pewno również tam są.

Jeżeli testujemy frontend z poziomu testów automatycznych, to mamy dwie najpopularniejsze biblioteki do testów, czyli Selenium. I narzędzie, które obecnie coraz bardziej się rozpycha na rynku, czyli Playwright. Oczywiście, skoro używamy przeglądarki, to w przeglądarce są jeszcze DevTools, które umożliwiają podgląd kodu HTML, jaka jest komunikacja i tak dalej. Więc z tymi narzędziami bardzo często tester automatyzujący aplikacje webowe ma styczność.

Kolejnym zestawem narzędzi są narzędzia do testowania wydajności. Mamy na przykład Gatlinga, K6 czy bardzo sławetny JMeter. W Pythonie, którego używam, jest biblioteka Locust, która jest oczywiście oparta o bibliotekę Request, o której przed chwilą mówiłem w przypadku zwykłych testów automatycznych REST API.

Mamy jeszcze testy urządzeń. Jeżeli testujemy jakieś urządzenie, to tutaj się pojawiają rzeczy związane z samą architekturą czy ze sprzętem. Może to być, nie wiem, oscyloskop, mogą to być jakieś karty wejścia wyjścia, które badają sygnały, wysyłają sygnały do urządzenia itd.

Jeżeli testujemy interfejs sieciowy danego urządzenia, czyli sprawdzamy, czy potrafi się komunikować po ethernecie, to mamy jakieś sniffery sieciowe, czy narzędzia typu Wireshark, które służą do analizy pakietów przesyłanych po sieci. W zależności od tego, co testujemy i

gdzie testujemy, te narzędzia mogą być bardzo podobne pomiędzy kilkoma grupami albo zupełnie inne. Wiedza, która tu jest potrzebna, znowu w zależności od tego, co robimy.

Idziemy coraz głębiej. Jakie są metryki i wskaźniki, które są używane do mierzenia skuteczności testów? Jak wygląda dokumentowanie całości? Tak mocno skrótowo.

Jest dużo różnych metryk i te, które ja tutaj wymienię, to jest tylko wycinek tego, co jest dostępne. Najczęściej widzianą metryką, z którą bardzo dużo developerów też ma styczność, to jest tak zwany code coverage, ta metryka mówi o tym, ile linijek kodu ma pokrycie w testach jednostkowych, czyli ile testów jednostkowych testuje linijki kodu.

Od razu zapytam, ile faktycznie te testy testują, a ile, wiesz, osiągamy procentów.

Jest dużo ciekawych aspektów. Taka ciekawostka w Pythonie. W Pythonie do badania code coverage jest biblioteka, która nazywa się coverage i ma jedno z ustawień, które według mnie powinno być domyślnie włączone, tak zwany branching. Biblioteka sprawdza, czy kod testów jednostkowych sprawdza odnogi w instrukcjach warunkowych, na przykład if-else. Domyślnie to ustawienie jest wyłączone, jeżeli stworzymy testy jednostkowe dla kodu, który jest napisany, a nie mamy włączonej tej opcji, to jesteśmy w stanie bardzo szybko dobić do 100% pokrycia jakiejś biblioteki, którą piszemy.

Wystarczy, że zmienimy to ustawienie z false na true i nagle pokrycie kodu spada nam do 70-80%. Jest dużo ciekawych aspektów z tym związanych, bo w zależności, jakie narzędzie używamy, teoretycznie

najlepiej byłoby, żeby było 100%. Ja mam troszeczkę wrażenie, że developerzy uznają, że nawet jeżeli mamy 100% pokrycia kodu testami jednostkowymi, to aplikacja jest dobrze działająca. A to nie jest prawda, bo jeszcze mamy te wszystkie inne poziomy testów, na przykład testy integracyjne, w których coś może nie działać, ale na poziomie testów jednostkowych wszystko mamy na zielono, 100% pokrycia kodu.

Trzeba mieć pewną świadomość, że nawet idealnie zrobione testy na jakimś poziomie nie uchronią nas przed błędami w całej aplikacji, która gdzieś tam jest stworzona czy wgrana na produkcję.

Podsumowałbym, że statystyka kłamie albo że to, co w tabelach widać niby wszystko ładnie, to w życiu wcale tak nie jest.

Jeszcze jest następny aspekt, że jeśli chodzi o sam code coverage, to ideałem byłoby to 100%, a tak naprawdę powinniśmy mieć ponad 100%, bo są sytuacje, w których możemy coś przetestować jeszcze na innym poziomie i zdublujemy testy tej samej funkcjonalności, ale w szerszym zakresie. Zamiast testowania jednej funkcji, to testujemy coś poziom wyżej, czyli funkcję, która używa tej funkcji. Powiedzmy, że ten code coverage powinien być tak naprawdę powyżej 100%, a prawda jest taka, że w większości projektów uznaje się, że 70-80% code coverage to jest dobry code coverage. Tutaj, tak jak mówiłem, to jest kolejny etap, na którym można by się było doktoryzować.

Przejdźmy do kolejnej metryki, która jest troszeczkę z tym powiązana, czyli test automation coverage. To jest bardziej już nie przez developerów, a przez testerów wykorzystywana metryka, która mówi o tym, ile mamy testów automatycznych do scenariuszy testów

manualnych. Czyli jeżeli mamy testera manualnego, który rozpisał 200 scenariuszy testów manualnych, to czy mamy wszystkie te scenariusze zautomatyzowane, wtedy mamy 100%. Podobnie jak w przypadku code coverage, nie zawsze się to udaje. Bardzo często jest tak, że tego 100% nigdy nie osiągniemy, bo są sytuacje, że pewnych scenariuszy testów manualnych nie jesteśmy w stanie zautomatyzować. Albo automatyzacja tego jest tak kosztowna albo tak długo by trwała, że się z tego rezygnuje i takie testy zawsze wykonuje się ręcznie.

Są jeszcze inne metryki, jak function feature coverage. Te metryki się stosuje w momencie, jeżeli mamy bardzo dobrze po stronie specyfikacji aplikacji rozpisane funkcjonalności, które mamy. Mamy na przykład, punkt 1, 3, dodanie produktu do koszyka. Jeżeli mamy w ten sposób rozpisaną całą funkcjonalność aplikacji, to możemy mieć 300 takich punktów. Tworząc scenariusz testu manualnego, czy automatyzującego, mówimy, że ten test automatyzuje nam funkcjonalność w tym i w tym punkcie. W ten sposób jesteśmy w stanie śledzić, ile takiej funkcjonalności mamy zautomatyzowane, czy są pokryte testami na poziomie testów manualnych, czy automatycznych.

Kolejną metryką, która często się pojawia, to jest na przykład returning defects, czyli ile błędów, które były wcześniej wykryte, znowu wraca do nas od klienta. Przykładowo mieliśmy taką sytuację, że w wersji 1.0 mieliśmy jakiś błąd, naprawiliśmy go w wersji 2.0, wgraliśmy tę aplikację na produkcję, mamy aplikację w wersji 3.0 kolejną i ten sam błąd, który był w wersji 1.0, znowu się pojawia. Teraz my, po stronie testów tego nie wykryliśmy, bo nie mieliśmy czasu na zautomatyzowanie testu, który wykrywał ten konkretny błąd. A on został wprowadzony jako regresja w

wersji 3.0 i klient znowu nam zgłasza ten sam defekt. Metryk po stronie testów może być naprawdę bardzo, bardzo dużo. My tutaj, to co ja powiedziałem, to jest tylko kilka najbardziej, powiedzmy, popularnych metryk, ale fantazję można mieć bardzo, bardzo dużą. Kolejny etap, że można by było się z tego doktoryzować i pisać książkę.

To może jeszcze dokumentacje, Powiedzmy coś na temat dokumentacji.

Tak, dokumentacja. Część rzeczy już mówiliśmy, czyli te scenariusze testów manualnych. Mówiliśmy o TestRailu czy Xray'u, no to jest bardzo częsta odpowiedzialność testera manualnego, czy ewentualnie w niektórych firmach testera automatyzującego. Dokumentacją może być kod testów automatycznych. Jeżeli coś automatyzujemy, to tak samo jak w przypadku developerów. Źródłem prawdy o tym, jak działa aplikacja, jest kod aplikacji. Tak samo jest po stronie testów. Źródłem prawdy o tym, co testy automatyczne testują, jest ich kod. Możemy pewne rzeczy do kodu wrzucić, nie tylko sam kod, ale możemy poprzez jakieś komentarze, opisy funkcji i tak dalej, pewne rzeczy dokładać do samego kodu i później w raporcie z testów już pewne rzeczy wyciągnąć.

Innym poziomem dokumentacji są chociażby zgłoszone błędy w Jira. One też mogą się odnosić na przykład do specyfikacji, mogą mówić nam o tym, jaki jest stan danej aplikacji. Czy dane błędy są naprawione, czy nie. To też jest wszystko jakimś poziomem dokumentacji, szczególnie jeżeli dochodzi do procesu certyfikacji firmware'u, o której wcześniej mówiłem, czy jakiegoś innego software'u. Takie statystyki o tym, ile jest błędów nienaprawionych i tak dalej, trzeba umieścić w takiej dokumentacji, żeby utrzymać certyfikację.

Innym typem dokumentu jest test plan, który opisuje cały proces testowania danej aplikacji. Tam opisujemy, na których etapach, które komponenty, jakie mają mieć wyniki końcowe, na jakich poziomach. Ilość zgłoszonych błędów, z jakim severity, czyli na jakim poziomie wpływu na aplikację dany błąd jest i tak dalej. Taki test plan powinien bardzo dużo rzeczy pokrywać. Nie we wszystkich firmach się je tworzy. Test plany raczej tworzy się w środowiskach korporacyjnych i takich mocno ustrukturyzowanych. W mniejszych firmach czy start-upach o test planie przez bardzo, bardzo długi okres czasu w ogóle się nie myśli. Tworzy się testy, a test plan ewentualnie na którymś poziomie powstaje, bo przyjdzie nowy szef albo ze zwykłego start-upu zaczynamy się przeistaczać w bardziej ustrukturyzowaną formę organizacji.

Taki najbardziej popularny dokument, to raport z przeprowadzonych testów. Czy to na poziomie testów manualnych będzie to jakiś Excel czy inna tabelka w której opiszemy, co się wydarzyło w teście i jaki był wynik końcowy. W przypadku testów automatycznych będzie to jakiś plik JUnit lub na Jenkinsie będzie lista testów ze statusem „przeszło” lub „nie przeszło” i tak dalej. W niektórych przypadkach, takich ustrukturyzowanych organizacji jak korporacje, są jeszcze zbiorcze raporty z testów. Ten dokument zbiera w całość inne dokumenty, czyli na przykład raporty z testów funkcjonalnych, raport z testów integracyjnych, raport z testów wydajnościowych, raport z certyfikacji i na końcu zbiera się to w taki zbiorczy dokument. Opisuje się to w odpowiedni sposób i taki dokument często jest częścią procesu oficjalnego release'u jakiegoś oprogramowania.

Oczywiście tych dokumentów może być dużo, dużo więcej. To są takie dokumenty, które w tym momencie przychodzą mi na myśl jako najczęściej spotykane.

Po tej całej dawce informacji czas zadać pytanie, jak wygląda typowy dzień pracy testera oprogramowania?

Trochę śmieszkując: kawa, meeting, maile, chwila pracy, raport z godzin, no i do domu. Ewentualnie jeszcze po drodze jakaś inna kawa. Czasem się zdarzają takie dni, że tak niestety to wygląda. Ten dzień będzie wyglądał inaczej w zależności od tego, w którym miejscu w organizacji jako tester jesteśmy albo jaką rolę pełnimy.

Tester manualny powinien być takim poziomem, który każdy przechodził albo gdzieś tam pracował. Tester manualny, aby stworzyć dobry scenariusz do testów, żeby miał nad czym pracować, po pierwsze musi mieć aplikację wgraną, a po drugie musi mieć jakiekolwiek informacje na temat, jak ta aplikacja ma działać. Ideałem byłoby, aby mieć specyfikację. Nie zawsze ta specyfikacja jest. Czasem są to zwykłe wpisy w jakimś narzędziu do planowania pracy typu Jira, gdzie są tickety związane z opisem funkcjonalności. Czasem te tickety są bardzo słabo opisane, więc wtedy jako tester musimy znaleźć kogoś, kto nam ten ticket bardziej rozpisze. Czy product owner, czy analityk biznesowy. W każdej firmie może to zupełnie inaczej wyglądać, tutaj też ciężko jest powiedzieć, że w danej firmie będzie to wyglądało tak, a u Ciebie inaczej. Wszystko zależy.

A jest tak, że jest jedna osoba, do której powinniśmy w pierwszej kolejności uderzyć?

Powinno tak być. Powinna być taka osoba wyznaczona w zespole, ale nie zawsze tak jest. Różne są sytuacje. Powinien być chociażby jakiś product owner, który wie, ma jakiś zamysł na to, jak ten produkt ma wyglądać, jak ma działać. Wtedy do takiej osoby powinniśmy się odwołać i albo ta osoba nam wytłumaczy albo właśnie uzupełni jakiś ticket albo wskaże nam dokument w którym jest to opisane. Czasem jest tak, że takim product ownerem jest na przykład jakiś developer albo ktoś, kto ma zupełnie inną rolę. Bywa to bardzo nieformalne, ale taka osoba musi być, bo ktoś musi powiedzieć, jak ta aplikacja ma działać. To się nie bierze z powietrza.

Wspomniałeś o developerze, ale wydaje mi się, że najgorszą opcją jest zapytanie osoby, która realizowała tę implementację, bo ona może mieć swoją koncepcję, która jest niezgodna z tą koncepcją, która ma być. Jeżeli tester zapyta tego programisty, to odpowie on, jak chciał to zrobić, a nie jak miało być zrobione.

Tak, to jest najgorsza opcja. To prawda, co mówisz. W idealnym środowisku mamy specyfikację. Teraz nawet jeżeli pójdziemy do developera i go zapytamy o coś i on powie, że jest tak, a w specyfikacji jest inaczej, zaczynamy wpadać w polemikę z nim i musimy znaleźć kogoś, kto podejmie decyzję, czyli jakiegoś product ownera. Nie zawsze tak jest. Czasem może być tak, że musimy coś na szybko sprawdzić, nie mamy kogo zapytać. Najbliższą osobą, która wie, jak to działa, jest ten developer, który to pisał. Trzeba być na tyle elastycznym i mieć wyczucie pewnych rzeczy, żeby wiedzieć, kogo w którym momencie zapytać o pewne rzeczy. Świat nie jest idealny, niestety.

Co tester jeszcze robi? Testuje. Jak już napisał scenariusz do testów, to musi przeklikać tę aplikację, stworzyć raporty z tych testów czy raporty z błędów, jeżeli jakieś znajdzie, umieścić to w Jira, odpowiednio opisać. Napisać, jaka jest reprodukcja tego błędu, co należy wykonać, żeby ten błąd się pojawił. Dobrze jest, jeżeli wrzuci jakiś screenshot do Jira, pokaże, że w tym miejscu w aplikacji, jak zrobił to i to, to pokazało się coś takiego, a powinno działać tak i tak, bo na przykład w specyfikacji w tym punkcie jest opisane, że to ma działać inaczej.

Jeżeli tester przeprowadza testy regresji albo pojawia się nowa wersja aplikacji, to musi sprawdzić, czy błędy, które developerzy twierdzą, że zostały naprawione, faktycznie zostały naprawione, bo może się zdarzyć, że ktoś coś przeoczył albo zrobił inaczej, niż było zakładane. Weryfikacja poprawek czy ponowne retesty aplikacji jako całość. No i to co mówiliśmy, czyli tworzenie scenariuszy jest tutaj potrzebne.

Tester automatyzujący jest rozwinięciem tych rzeczy, o których mówiliśmy. Może mieć konieczność przeprowadzenia tych samych kroków co tester manualny, ale jeżeli automatyzujemy testy, to głównym zadaniem testera automatyzującego jest pisanie kodu testów. Bierze scenariusz, jeżeli go ma, tworzy kod testu automatycznego. Bardzo często w firmach testy automatyczne są stworzone w ten sposób, że o jakiejś godzinie, często po godzinach pracy 18-20, wszystkie testy automatyczne, które już są stworzone, są uruchamiane na jakimś środowisku testowym i sprawdza się, czy codziennie te wyniki testów są takie same.

Może się zdarzyć, że jest jakiś błąd w aplikacji, który codziennie się pojawia i wiemy, że ten test nie działa, bo jest błąd. Ale może się też

zdarzyć coś takiego, że na przykład co drugi dzień jakiś jeden test często jest na zielono, ale co trzeci dzień jest na czerwono, bo nie działa. Tu się zaczyna cała zabawa z tym związana. Po pierwsze taki tester automatyzujący musi sprawdzić, czy to jest wina testu, bo może być test w jakiś dziwny sposób napisany, że z jakichś powodów czasem działa, a czasem nie działa. Może jest jakiś problem w aplikacji i z jakichś przyczyn w którymś momencie jest jakieś opóźnienie w sieci albo aplikacja w danym momencie tworzy jakiś złożony raport i nie ma na tyle zasobów, żeby zwykłą czynność wykonać. Trzeba zrobić dochodzenie, dlaczego dany test jest czasem na zielono, a czasem na czerwono.

Jeżeli tworzymy kod testów, to proces jest bardzo podobny jak w przypadku programisty. Jeżeli mamy trochę większy zespół albo mamy jednego testera automatyzującego i programistów, fajnie jest zrobić code review tego, co tester zrobił. Albo robimy to pomiędzy członkami zespołu testerskiego albo robią to programiści, jeżeli jest taka możliwość.

Kolejny aspekt, który już częściowo opisałem, to cały proces CI/CD, czyli tworzenie pipeline'ów, które będą uruchamiały testy w nocy, czy w momencie jakiegoś deploymentu na jakieś środowisko testowe czy produkcyjne. Możemy to połączyć ze smoke testami, o których mówiliśmy, że mamy małą suitę testów, które jeżeli pójdzie deploy na produkcję, to automatycznie CI/CD uruchomi smoke testy na produkcji. Później mamy raport i widzimy, czy działa, czy nie działa.

Co jeszcze? Są aspekty związane z samoutomatyzacją, na przykład trzeba czasem zgłosić albo poprosić developerów, żeby wprowadzili tak zwane test ID w HTML, które ułatwi nam tworzenie testów w aplikacji webowej. Albo musimy uzyskać jakieś dostępy do środowiska na jakimś

poziomie, bo użytkownik normalnie pewnych rzeczy nie może zrobić ze względu na bezpieczeństwo. Ale my żeby przeprowadzić testy automatyczne, to musimy mieć możliwość dogadania się lub rozmowy z jakimś serwerem, do którego normalny użytkownik nie ma dostępu. Są więc takie rzeczy związane właśnie z całą infrastrukturą do testów, które są bardzo często potrzebne.

Wróciłbym do tych nocnych testów. Czy one w takiej dużej aplikacji trwają godzinę, czy mogą trwać dłużej? Często się mówi, że im szybciej, tym lepiej. Zastanawiam się, jak to wygląda od strony takich testów automatyzujących całość. Czy to trwa bardzo długo i bada się, ile zaoszczędziliśmy czasu na tych testach automatycznych w porównaniu do testów manualnych?

To jest dobre pytanie. Mamy tę słynną piramidę testów. Wyobraźcie sobie ten kształt piramidy – najszersza jest u podstawy, a u podstawy mamy testy jednostkowe, czyli to, co tworzą programiści. Te testy są napisane w taki sposób, że przeprowadzenie ich trwa w porywach kilka sekund. Czasem przygotowanie jakiejś struktury może trwać kilka minut. Im wyżej w piramidzie, gdzie zaczyna się ona zwężać, tym więcej pojawia się testów, które są kosztowne czasowo. Na samym czubku tej piramidy w większości przypadków są testy związane z interfejsem użytkownika, czyli tutaj mamy jakieś Selenium czy Playwright i przeglądarkę.

Ile te testy trwają? Tak jak powiedziałeś, im krócej, tym lepiej. Ale jeżeli mamy złożoną aplikację pod kątem funkcjonalności, z wieloma funkcjonalnościami, to przeprowadzenie wszystkich testów może trwać nawet kilka czy kilkanaście godzin. Jak się takie problemy z czasem

rozwiązuje? Po pierwsze, testy często dzieli się na tzw. mniejsze suity, a jedną z takich suit są smoke testy. To jest wycinek testów, które sprawdzają najbardziej kluczowe funkcjonalności. Tych testów powinno być kilka, kilkanaście, po to, żeby były szybkie. Po co szybkie? Po to, że jeżeli developer wgrywa hotfixa lub pełną nową wersję dla klienta końcowego na produkcję, to przeprowadzenie testów, które trwają 8 godzin zaraz po deploymencie, niekoniecznie ma sens. Te testy powinny być wcześniej uruchomione, żeby upewnić się, że to, co chcemy wgrać, jest dobre. Po wgraniu na produkcję chcemy się upewnić, że deployment poszedł dobrze, więc uruchamiamy kilka testów, które trwają w porywach 10-15 minut.

Jednym ze sposobów na skrócenie czasu wykonywania testów jest zrównoleglenie testów, czyli możemy uruchomić kilka, kilkanaście testów jednocześnie. Trzeba to jednak umieć dobrze zrobić, bo teoretycznie testy powinny być od siebie niezależne, co nie zawsze się udaje. Trzeba mieć pewne wyczucie.

Ile to trwa? To zależy ile mamy testów, jak rozbudowana jest aplikacja. W jednej z firm, w której pracowałem, testowaliśmy urządzenia, a testy trwały średnio 8 godzin. Były one uruchamiane codziennie po godzinie 17, trwały do którejś w nocy, a rano przychodziliśmy, mieliśmy wyniki. Problem pojawił się, gdy weszły nowe urządzenia, które umożliwiały kilka różnych konfiguracji. Uruchomienie testów dla każdej konfiguracji trwało 8 godzin, a konfiguracji mieliśmy pięć. Uruchomienie wszystkich możliwych konfiguracji jednego dnia trwałoby 40 godzin. Nikt nie mógłby napisać nowego testu, bo nie miałby dostępu do urządzeń, pojawiałyby się inne problemy.

Jak to rozwiązaliśmy? Po pierwsze, poszła prosta prośba: *potrzebujemy więcej środowisk testowych*, żeby na jednym środowisku była pierwsza konfiguracja, na drugim druga itd. Możemy też dzielić testy na mniejsze kawałki, czyli jednego dnia puszczamy połowę testów w pierwszej konfiguracji, drugiego dnia drugą połowę w tej samej konfiguracji. W zależności od tego, co robimy, gdzie jesteśmy i co chcemy osiągnąć, trzeba radzić sobie w jakiś sposób, aby zoptymalizować czas pod wieloma względami.

Teraz chyba najważniejsze pytanie z całego podcastu: od czego powinniśmy zacząć, jeśli interesuje nas praca testera oprogramowania i ile powinniśmy umieć, aby starać się o pierwszą pracę?

Znowu zażartuję: powinniśmy umieć wszystko. Trochę śmieszkując, ale to też gorzka prawda. 15 lat temu, kiedy zaczynałem, praca testera była dużo prostsza, a testerzy często mieli złą renomę. Wracając do tego, co wcześniej wspomniałem, kontakt między testerem a developerem był bardzo zaburzony. 15 lat temu, żeby zostać testerem, wystarczyło umieć obsłużyć Excela i wyklikać coś w przeglądarce. To w większości przypadków wystarczało na początku drogi. Obecnie próg wejścia do branży IT na poziomie testera jest dużo wyższy, ponieważ poziom skomplikowania aplikacji jest o wiele większy. Często się w sieci pojawia, że wszystko, co proste, zostało już napisane, więc aby teraz dobrze testować czy wejść do tej branży, musimy wiedzieć dużo więcej. Poziom juniora dziś to coś, co 10-15 lat temu było poziomem mida jako tester.

Co musimy umieć? Musimy zaznajomić się ze słownictwem, które jest używane. Nawet te zwroty, które użyłem dziś podczas naszej rozmowy, są opisane w większości przypadków i uwspólnione po to, aby mieć wspólny język. Pierwszym miejscem, do którego bym zajrzał jako osoba chcąca wejść do branży, jest sylabus ISTQB. To dosyć sławny certyfikat dla testerów, który robi wiele osób. Mam mieszane uczucia co do tego certyfikatu, bo uważam, że ideałem byłoby, aby ktoś popracował chociaż pół roku, a potem robił ten certyfikat. Niestety, z powodu rosnącego poziomu wejścia, wiele osób robi ten certyfikat, aby podnieść swoje kwalifikacje. Moje zdanie jest takie: zanim zrobimy certyfikat, przeczytajmy sylabus kilka razy, spróbujmy go choć trochę zrozumieć i nauczyć się słownictwa, które tam jest.

Kolejnym etapem, który powinniśmy przejść na etapie zastanawiania się nad wejściem do branży, jest przeczytanie i poszukanie ogólnodostępnych, darmowych materiałów w internecie. Tych materiałów jest naprawdę bardzo dużo. Na przykład Twój podcast, gdzie omawiasz wiele różnych ciekawych rzeczy, czy to z testowaniem, czy z samym procesem tworzenia aplikacji, developmentu itd. Masz wielu ciekawych gości, którzy opowiadają o różnych interesujących kwestiach. Można się osłuchać z pewnymi rzeczami. Te rzeczy, które tutaj goście omawiają, można wyszukać w Google czy w innych przeglądarkach, zainteresować się tym, z czym mamy do czynienia.

Mamy też wiele blogów dla testerów. Jednym z blogów jest mój testerembyc.pl. Nie ma tam bardzo dużo materiałów, bo nie mam aż tak dużo czasu, żeby go rozwijać, ale jest na przykład blog wyszkolewas.com.pl prowadzony przez Waldka Szafrańca, gdzie jest

naprawdę bardzo dużo materiałów na poziomie entry level, ponieważ Waldek szkoli przyszłych testerów jako mentor.

Mamy również wiele podcastów dla testerów. Jednym z nich jest podcast: *Po szklanie i na testowanie*, który prowadzą Jakub Konicki i Paweł Kowalczyk. Właśnie zaczyna się trzeci sezon, w którym chłopaki omawiają wiele różnych ciekawych rzeczy związanych z testowaniem, zarówno na poziomie podstawowym, jak i trochę wyższym. Bardzo fajnie się ich słucha, polecam. Jest też podcast, do którego przychodzą różni goście, prowadzony przez Norberta Jankowskiego: *Tester oprogramowania podcast*. Tam również można znaleźć wiele ciekawych informacji.

Mamy także kanały na YouTube, które opisują pracę testera. Zdecydowanie polecam kanał Kuby Rosińskiego, który nazywa się: *TestITka*. Osoby zapraszane do tego kanału dostają zadanie testera. Muszą przeprowadzić testy eksploracyjne jakiejś aplikacji, której nigdy wcześniej nie widziały i mają na to 40 minut. To bardzo fajny wgląd w to, jak może wyglądać praca testera, zdecydowanie polecam. Jest też kanał Bartka Kity: *AkademiaQA*, gdzie również są ciekawe informacje związane z testowaniem.

Oczywiście mamy również książki. Najbardziej znaną książką w świecie testerów jest: *Zawód Tester* Radosława Smilgina, która opisuje dużo aspektów związanych z pracą testera: jak ona wygląda, co trzeba umieć. Tutaj jedna uwaga: ta książka ma już kilka lat, nie jest jakaś mega aktualna pod względem tego, co obecnie się dzieje na rynku, ale jest to bardzo dobry wstęp do ścieżki testera, żeby zaznajomić się z tym, co się dzieje w branży, jak to powinno wyglądać i co trzeba umieć.

Troszeczkę bardziej aktualna książka napisana półtora roku temu mniej więcej, miałem przyjemność recenzowania tej książki i jest to książka *Jakość to będzie!* Waldka Szafrańca, z bloga, o którym wcześniej tutaj mówiłem, też polecam.

Mamy grupy na Facebooku, gdzie jest bardzo dużo materiałów i ciekawych rozmów. Grupa, która w tym momencie nie pamiętam, ale chyba powyżej 50 tysięcy osób zrzesza z całej Polski, jest to grupa: *Testowanie oprogramowania*. Dla osób wchodzących do branży, czy chcących wejść do branży od strony testera, jest grupa: *Testowanie oprogramowania wsparcie na starcie*.

Miejsc, gdzie można zacząć zaznajamiać się z tym, jest bardzo dużo i zanim kupicie jakikolwiek kurs, szkolenie i tak dalej. Zapoznajcie się z tymi materiałami, czy to jest coś, co może was interesować.

Jeszcze jest jedno fajne miejsce. W grupie: *Testowanie oprogramowania* administratorem jest Piotrek Wicherski. Tam jest gdzieś podpięty zbiór materiałów, które on tworzy. On zbiera tak naprawdę linki, różne informacje dotyczące testowania oprogramowania, więc również warto tam zajrzeć.

Co jeszcze? Na pewno to, o czym już gdzieś tam na początku wspomnieliśmy, czyli możliwość uczestniczenia w lokalnych meetupach. Jest tych meetupów w Polsce stricte dla testerów kilka. W jednym z nich, ja jestem współorganizatorem, o czym wspomniałem, czyli Ślonzacz-QA tutaj na Śląsku w Katowicach. Wskrzesiła się przed wakacjami ŁódQA w Łodzi. Jest jeszcze 3QA w Trójmieście, ŁuczniczQA w Bydgoszczy oraz KraQA w Krakowie.

Tam w tych wszystkich nazwach jest QA? Żeby było jasne.

Tak, QA na końcu, dokładnie.

Jest jeszcze WarszawQA, tylko niestety WarszawQA najprawdopodobniej zawiesiła swoją działalność. Była też WrotQA we Wrocławiu, która chyba też obecnie nie ma planów na wskrzeszenie, ale warto obserwować na grupie: *Testowanie oprogramowania* na Facebooku, bo tam przeważnie są informacje kiedy, kolejne jakieś spotkanie się pojawi.

Są też konferencje. Niektóre są darmowe, na przykład: *Na Podbój IT* w Bydgoszczy. I o ile dobrze pamiętam, to: *Test Dive* w Krakowie. Jest też: *test:fest* we Wrocławiu, przy czym tutaj trzeba czasem polować powiedzmy na te darmowe bilety albo dobrze uzasadnić, dlaczego chcecie mieć wstęp na taką konferencję. Są też płatne konferencje, ale na poziomie osoby, która chce wejść do branży, uważam, że nie do końca warto, bo jeżeli ktoś ma wydać dwa tysiące złotych na bilet wstępu, to lepiej sobie przeznaczyć to na przykład na lekcję z mentorem jakim jest Władek Szafraniec czy Norbert Jankowski, którzy Wam pokażą już od takiej bardziej praktycznej strony, jak do tego wejść i czego się nauczyć.

Co jeszcze? Bardzo dużo podczas naszej rozmowy starałem się podkreślić, co robi tester manualny, czyli tworzy te scenariusze testów, więc my jako osoba, która chce wejść do branży i chce się uczyć, to trzeba się nauczyć tworzyć te scenariusze testów. Tutaj materiały też są dostępne w internecie. Bardzo dużo osób, które wchodzą do tej branży, tworzą sobie na GitHubie swoje portfolio, w którym umieszczają

dokumenty z takimi scenariuszami czy jakiś prosty kod testów automatycznych. Tutaj dochodzimy do kolejnego etapu, czyli nauczenia się programowania i tworzenia jakichś prostych testów automatycznych na przykład z wykorzystaniem Playwright, który jest obecnie chyba najbardziej poszukiwanym w ofertach pracy. Nauczyć się programować, niekoniecznie samej biblioteki konkretnej, bo to przyjdzie z czasem, ale jeżeli myślicie o pracy jako tester i chcecie automatyzować, poświęćcie sporo czasu na naukę programowania, nie od razu automatyzacja, tylko programowanie. Nauczcie się, jakie są struktury danych, jakie są instrukcje sterujące, czym są klasy i tak dalej, bo to się Wam w pracy przyda. Jeśli pominiecie ten krok, to naprawdę wasza nauka będzie dużo trudniejsza. Nauka podstaw programowania jest czymś, co bardzo polecam.

Kolejnym etapem, to też związane z automatyzacją, jak działa i jak testować REST API. Można tutaj zacząć od tego narzędzia Postman, ale jeżeli chcecie automatyzować, to również od strony kodu warto wiedzieć, jak wysyłać requesty, jak odbierać, jak te dane, które tam są umieszczać czy wysyłać, pobierać, jak patrzeć, jakie są kody odpowiedzi, czym jest 200, czym jest 404 i tak dalej.

Coś, czym warto się też ewentualnie zainteresować właśnie od strony automatyzacji, może być język SQL. To jest język do rozmawiania z bazą danych. W ofertach może nie jest on dosyć często obecnie spotykany, bo często w projektach są już wykorzystywane bazy NoSQL. Tam troszeczkę inaczej się z taką bazą rozmawia. Ale nauka SQL, przynajmniej podstaw jakichś, jak napisać zapytanie SELECT, INSERT, UPDATE i tak dalej, na pewno nie będzie wiedzą straconą. Tym bardziej,

że to nie jest coś mega skomplikowanego. Tych podstaw można się nauczyć w kilka dni.

Coś, co bardzo dużo osób wchodzących do branży próbuje robić, ja tego nigdy nie robiłem, czyli tak zwany Crowd Testing. Jednym z takich najbardziej popularnych portali, które coś takiego oferuje, jest portal test.io. To jest taki wstęp do testowania na żywym projekcie. Są firmy, które szukają testerów, którzy przetestują jakąś aplikację i można tam na tej stronie po prostu aplikować o taką powiedzmy pracę w jakimś projekcie. Jeżeli znajdziecie jakieś fajne błędy, odpowiednio je udokumentujecie, to możecie jakieś drobne pieniążki za to dostać. Ja tego nigdy nie robiłem, bo w moich czasach tego nie było. Wiem, że dosyć sporo osób, które chcą wejść do branży, próbują w tym swoich sił, nawet nie dla zarobków, ale styczności z realnym projektem i sprawdzenia się, czy jesteśmy w stanie w odpowiedni sposób coś przetestować, jak to wygląda w projekcie, jak zaraportować błędy i tego typu rzeczy.

To zróbmy jeszcze małe podsumowanie. Jakie zmiany zaobserwowałeś w branży w okresie ostatnich lat? Jakie przewidujesz w najbliższych latach, w dobie popularyzacji AI?

Zdecydowanie dwie rzeczy wychodzą takie bardzo widoczne. Po pierwsze, na tej przestrzeni tych piętnastu lat na pewno bardzo podniósł się próg wejścia do samej branży jako tester. Musimy wiedzieć dużo więcej niż jeszcze te dziesięć czy piętnaście lat temu, żeby w ogóle ubiegać się o rolę tego juniora.

Na pewno też widać dużo większe zainteresowanie pracą w branży IT od strony testów. Ten obraz został tak trochę fałszywie nakreślony przez wszelkiego rodzaju bootcampy, które reklamują, że wejście do branży IT od strony testów jest najprostsze. Bootcampy żyją z tego, że sprzedają szkolenia. Przez to i też pandemię i tak dalej widać, że bardzo dużo ludzi ciągnie do tej branży IT. Tutaj branża była dosyć stabilna. W pandemii zarobki były dosyć dobre. A to też jest taki trochę mylny obraz, bo te zarobki są dobre, ale dla ludzi, którzy coś umieją, a niekoniecznie dla ludzi, którzy dopiero wchodzą do tej branży. Można się zdziwić, jeżeli ktoś ma, powiedzmy trzydzieści, czterdzieści lat, dobre stanowisko w innej firmie i zarabia w miarę sensowne pieniądze, a usłyszał o tym, że w IT to się zarabia po dwadzieścia tysięcy na rękę. Może bardzo mocno zderzyć się ze ścianą w momencie, jak dostanie pierwszą ofertę jako junior, bo może się okazać, że zarobi dużo mniej niż w obecnej pracy nie związanej z IT.

Poziom skomplikowania samego oprogramowania jest dużo większy niż piętnaście lat temu. Kiedyś wystarczyła przeglądarka, czy aplikacja była napisana w PHP, z tyłu była jakaś baza danych MySQL, no i aplikacja była naprawdę bardzo prosta.

Obecnie do tego dochodzą całe rzeczy związane z tym CI/CD, pojęcia typu Infrastructure as a Code, czyli implementacja środowiska serwerów w chmurze, definiowania tego poprzez kod, bazy SQL, NoSQL, jakieś Redis Cache. Tych aspektów związanych z prostą, wydawałoby się, aplikacją webową jest dużo więcej niż było wcześniej. To też wymusiło podniesienie tego progu i poziomu wejścia.

Jeżeli chodzi o samą AI, to ja mam takie wrażenie, że obraz kreślony przez twórców samego AI jest trochę fałszywy. Z jednej strony wszyscy mówią, że nie uczcie się programowania, bo za parę lat to AI będzie programował za was i w ogóle. Prawda jest taka, że z punktu widzenia chociażby samych testów, po pierwsze ktoś będzie musiał testować to co AI stworzy, o ile to stworzy to w jakikolwiek sensowny sposób. Po drugie, żeby testować AI, wasz poziom wiedzy musi być dobry. Wy musicie umieć zweryfikować, czy to, co AI napisał, czy stworzył, jest dobre, poprawne i będzie działało. Musicie umieć to skontrolować, umieć ewentualnie powiedzieć AI, że w tym i w tym miejscu to powinno być zrobione inaczej. To znowu wymusi w pewien sposób podniesienie po raz kolejny tego progu wejścia do branży IT.

To, czego się z jednej strony obawiam, a z drugiej strony zawsze gdzieś tam było, ale AI to tutaj bardzo podkreśli, to zwiększenie przepaści pomiędzy osobami doświadczonymi, seniorami, a osobami wchodzącymi do branży. Jeżeli ktoś umie programować, czy jest dobrym testerem, czy testerem automatyzującym i wykorzysta AI jako narzędzie wspomagające jego pracę, będzie umiał je skontrolować, powiedzieć gdzie AI popełnił błąd i co powinien zrobić lepiej albo inaczej, to taka osoba będzie bardzo wydajna i bardzo poszukiwana na rynku pracy. Z drugiej strony junior, który będzie chciał wejść do branży IT, nie będzie miał tych wszystkich umiejętności seniorskich, bo to trzeba jednak swój czas poświęcić. Na naukę, obserwowanie tego, co się w projektach dzieje. Doświadczenie jest też potrzebne. Junior tego nie będzie miał, więc ta przepaść pomiędzy poziomem seniora, a poziomem wchodzącego do branży juniora jeszcze się zwiększy.

Ja troszeczkę przewiduję, że taki hype, który obecnie jest na pracę w branży IT w pewnym momencie się zastopuje, bo ludzie zdadzą sobie sprawę, że żeby wejść do tej branży IT trzeba będzie poświęcić bardzo dużo czasu, żeby mieć w ogóle szansę. Teraz bardzo dużo osób niestety nie ma tej świadomości.

Jeszcze dodam na koniec, pamiętajmy, że seniorzy nie rosną na drzewach. Ci juniorzy skądś się muszą brać, więc oni muszą nabrać tego doświadczenia. Firmy zrozumieją, że muszą zainwestować, żeby potem móc korzystać z dobrodziejstw takiego seniora. Seniora, który musi się najpierw nauczyć, musi być juniorem, midem czy tam regularem, a dopiero potem seniorem. Innej drogi nie widzę.

Nie ma innej drogi, ale tutaj jest problem, o którym chyba przed podcastem rozmawialiśmy, czyli moim zdaniem osoby decyzyjne muszą zrozumieć, że AI nie jest remedium na całe zło i ono nie poprawi samo z siebie wydajności programistów i nie zastąpi programistów. Niestety dużo osób które jest związane z AI, chociażby CEO NVIDII, cały czas mówią, że AI zastąpi programistów i niestety bardzo wielu managerów mocno w to wierzy, a rzeczywistość niestety jest troszeczkę inna.

Jak wspomniałeś o właścicielu, to ja dodam, że pamiętajmy, że on ma cel, on faktycznie na tym zarabia.

Tak, zarabia na łopatach. Bo jest gorączka złota, więc zarabia na łopatach.

Tak jest, on też może mówić to, co faktycznie mu odpowiada, a nie co faktycznie może myśli. Ciężko powiedzieć.

Mam pytanie, na które już odpowiedziałeś, więc może tylko tytuł i autorzy. Jaką książkę polecisz osobom, które chcą zostać testerami oprogramowania?

Gdybym miał polecić tylko jedną z tych dwóch książek, o których wspomniałem, to wydaje mi się, że poleciłbym książkę jednak Waldka Szafrańca: *Jakość to będzie!*, z tego względu, że jest po prostu świeższa. I wiele aspektów, które są opisane w obydwu książkach z perspektywy tego, co Waldek opisuje, jest bardziej zbliżone do obecnej rzeczywistości.

To na sam koniec, gdzie możemy Cię znaleźć w sieci, jeżeli ktoś by się chciał coś podpytać, coś poradzić, to gdzie ma Cię szukać?

Podstawowe miejsce to jest mój blog, czyli testerembyc.pl. Tam są też linki do chyba każdego innego medium społecznościowego, bo zarówno na Facebooku jest mój profil, czy prywatny, czy profil testerembyc.pl - Maciej Kusz. Na LinkedInie w obydwu również aspektach, przy czym bardziej prywatnie. Pod moim imieniem, nazwiskiem jako Maciej Kusz możecie mnie znaleźć. Mam też swój kanał na YouTubie, jest tam kilka filmów. Zresztą jak wpiszecie w YouTube'a Maciej Kusz, to myślę, że Wam wyskoczę nie tylko na swoim kanale, ale kilku innych, bo tak jak u Ciebie dzisiaj mam przyjemność występować, tak i u kilku innych osób byłem. Jestem też gdzieś tam na Instagramie, ale bardzo rzadko tam zaglądam, więc raczej myślę, że Facebook, LinkedIn i ewentualnie przez mojego bloga.

Zapraszamy wszystkich do odwiedzin miejsc, o których wspomniał Maciej. A ja Tobie, Macieju, bardzo dziękuję za tę rozmowę i podzielenie się z nami swoją wiedzą.

Ja Tobie również dziękuję za zaproszenie i wszystkim wchodzącym, czy chcącym wejść do branży życzę powodzenia.