PROGRAMIŚCI DLA SZCZECINA

Zachodniopomorska Szkoła Biznesu w Szczecinie

Teoria testowania oprogramowania

Część 3 – Zasady testowania, estymowanie, plan i strategia testów, CI/CD, metryki, narzędzia, ryzyka, pytania na rozmowie kwalifikacyjnej

Mariusz Łazor / 27.08.2019

Fundusze Europejskie Program Regionalny
Rzeczpospolita Polska
Pomorze Zachodnie
Unia Europejska Europejski Fundusz Społeczny

Zasady testowania

- Testowanie ujawnia usterki
- Testowanie gruntowne jest niewykonalne
- Wczesne testowanie
- Kumulowanie się defektów
- Paradoks pestycydów
- Testowanie zależy od kontekstu
- Mylne przekonanie o braku defektów

Testowanie ujawnia usterki

- Testowanie może służyć do wykazania, że w systemie istnieją defekty
- Testowanie nie jest w stanie dowieść, że w oprogramowaniu nie ma defektów
- Testowanie zmniejsza prawdopodobieństwo, że oprogramowanie będzie posiadało niewykryte defekty
- Jeżeli nie zostały znalezione żadne defekty, to nie jest to dowodem, że oprogramowanie ich nie posiada

Testowanie gruntowne jest niewykonalne

- Przetestowanie wszystkich możliwych scenariuszy (na podstawie różnych danych wejściowych i warunków początkowych) zazwyczaj jest niewykonalne – wynika to z działania w pewny stopniu niepewności i ograniczonych zasobów (czasu i budżetu)
- Im mniej skomplikowany system, tym dokładniej jesteśmy w stanie go przetestować
- Projektowanie testów jest sterowane analizą ryzyka i priorytetyzacją

Wczesne testowanie

- Im wcześniej rozpoczniemy aktywności związane z testowaniem, tym szybciej jesteśmy w stanie wykryć defekty
- Wczesne testowanie ogranicza ryzyko dostarczenia niezweryfikowanego systemu
- Łatwa identyfikacja źródła problemu

Kumulowanie się defektów

- Najczęściej najwięcej defektów znajdujemy w miejscach, w których wykonujemy najwięcej testów
- Z odpowiednimi umiejętnościami i doświadczeniem jesteśmy w stanie przewidzieć, w którym miejscu możemy się spodziewać największej ilości usterek
- Jedna usterka może spowodować wystąpienie innych (efekt „domina")

Paradoks pestycydów

- Powtarzanie tych samych testów może w końcu przestać ujawniać usterki
- Regularny przegląd i aktualizacja testów zwiększa ich skuteczność (są w stanie wykryć potencjalnie więcej defektów)
- Tworzenie nowych przypadków testowych pozytywnie wpływa na jakość

Testowanie zależy od kontekstu

Podejście do testów zależy od:

- Rodzaju aplikacji (inaczej testujemy np. aplikacje internetowe i mobilne)
- Skomplikowania systemu (ilości funkcjonalności, użytych technologii, architektury)
- Rodzaju testu

Mylne przekonanie o braku defektów

- Wykrycie i naprawienie dużej ilości defektów nie zawsze świadczy o poprawności oprogramowania
- System powinien przede wszystkim spełniać oczekiwania jego docelowych użytkowników – jeżeli ten warunek nie jest spełniony, to znaczy, że produkt nie jest prawidłowy

Estymacja testów – definicja wg ISTQB

„Obliczona aproksymacja wyniku związana z różnymi aspektami testowania (takimi jak wysiłek, data zakończenia, poniesione koszty, ilość przypadków testowych itp.), która jest użyteczna, nawet gdy dane wejściowe są niekompletne, niepewne lub zakłócone.”

Estymacja testów – co szacujemy?

- Pracochłonność procesu testowego – koszt wyrażony w roboczogodzinach lub roboczodniach wszystkich aktywności testowych (planowanie, realizacja, naprawa defektów, testy automatyczne itd.) - np. testowanie będzie kosztowało 15 roboczodni
- Czas trwania – wyrażenie pracochłonności na osi czasu uwzględniając takie czynniki jak dni wolne, nieobecności – np. testowanie będzie trwało od 1 do 30 czerwca
- Budżet – oprócz kosztu pracochłonności doliczane są również takie elementy jak licencje, sprzęt, koszty związane z delegacjami, szkolenia wewnętrzne, zewnętrzne
- Bardzo rzadko jesteśmy w stanie dokładnie oszacować powyższe wartości, dlatego w praktyce zakłada się jeszcze jakiś bufor, np. +15%

Co wpływa na estymację

- Typ umowy – dla umowy „fixed-price" (gdzie na samym początku spisywana jest umowa zawierająca ostateczny koszt projektu) konieczne jest możliwie dokładne oszacowanie kosztów, dla umowy „time and material" opracowujemy prawdopodobny koszt ustalając np. jego górną granicę, ale musimy raportować co zajęło ile czasu, żeby klient wiedział za co płaci
- Zakres i czas trwania projektu – ilość dostarczanych funkcjonalności, poziom skomplikowania
- Użyta metodyka wytwarzania – czy szacujemy całościowy budżet projektu (model kaskadowy), czy np. konkretnej iteracji (model iteracyjny)
- Zespół – ilość ludzi w zespole developerskim, doświadczenie, kompetencje techniczne, znajomość danej domeny biznesowej

Aktywności testowe podlegające estymacji

- Planowanie testów (zarządzanie procesem testowym, czas poświęcony na estymację również jest wliczany do kosztu)
- Przygotowanie testów (scenariuszy i przypadków testowych, środowiska, danych testowych)
- Raportowanie usterek i retesty – ile czasu zajmie zgłoszenie pojedynczego defektu i jego ponowne przetestowanie
- Testy regresywne
- Testy automatyczne
- Wdrożenie systemu

Techniki estymacji

- Wiedza ekspercka – szacowanie testów zajmuje się osoba z dużym doświadczeniem
- Dane historyczne – oparcie estymacji na koszcie podobnego projektu realizowanego w przeszłości
- W oparciu o parametry projektu – koszt testów jest obliczany ze wzoru, np. na podstawie ilości funkcjonalności czy jako współczynnik czasu poświęconego na implementację
- Estymacja trzypunktowa – szacujemy scenariusze pesymistyczne, średnie, optymistyczne i obliczamy średnią (arytmetyczną lub ważoną)
- Estymacja bottom-up – wyodrębnienie i estymacja niskopoziomowych zadań w projekcie i zsumowanie ich szacunków
- Spotkania, głosowania – interaktywne spotkania, na których cały zespół uczestniczy w szacowaniu kosztu

Strategia testów

- Wysokopoziomowy dokument określający ogólne podejście do testowania (oparcie się na modelach, specyfikacjach, testy eksploracyjne itd.)
- Skupia się na celu, zakresie z biznesowego punktu widzenia, procedurach (np. projektowanie przypadków testowych, zgłaszanie defektów), rolach, odpowiedzialności w procesie, metrykach
- Strategia jest sterowana wymaganiami biznesowymi (wysokopoziomowymi potrzebami)
- Dokument statyczny (rzadko się go zmienia)
- Może dotyczyć projektu lub całej organizacji
- Niejednokrotnie strategia jest włączana w plan testów

Plan testów

- Skupia się na tym co będzie testowane i w jaki sposób oraz wskazuje na to, co znajduje się poza zakresem testów
- Definiuje na jakich poziomach będą wykonywane testy (w kontekście konkretnego systemu)
- Jakie narzędzia zostaną użyte
- Opisuje podejście do testów automatycznych
- Specyfikuje środowisko testowe, które zostanie użyte
- Zawiera rozmieszczenie testów w czasie oraz estymaty
- Identyfikuje ryzyka
- Podstawą planu testów jest specyfikacja wymagań
- Jest powiązane z konkretnym projektem
- Dozwolona jest zmiana planu
- Czasami w planie testów opisywana jest również strategia testów

Scenariusz testowy

- Wysokopoziomowe określenie tego, co będziemy testować
- Jeden scenariusz skupia się na pojedynczym wymaganiu/module (ogólnie opisuje, co należy przetestować)
- Pokrywa wymagania biznesowe
- Są krótkie i zwięzłe (tzw. „oneliner”)
- Składa się z przypadków testowych

Przypadek testowy

- Lista aktywności wykonywanych dla danego scenariusza (jeden scenariusz posiada wiele przypadków)
- Przypadki testowe dokładnie opisują dany scenariusz
- Dokładnie opisuje co i jak należy przetestować – symuluje prawdziwe korzystanie z systemu
- Defekty są powiązane z przypadkami testowymi
- Zawierają tytuł, opis (wskazujący na cel), warunki wejściowe, kroki do realizacji wraz ze wskazaniem oczekiwanego zachowania na danym kroku, użyte dane wejściowe (jeżeli konieczne jest ich wskazanie)
- Dobrą praktyką jest powiązanie defektu z konkretnym przypadkiem testowym

Ciągła integracja (Continuous Integration)

- Praktyka zakładająca regularne umieszczanie wyników swojej pracy (najczęściej kodu źródłowego) w repozytorium projektowym
- Zakłada również regularną kompilację kodu źródłowego (np. automatycznie po każdej dodanej zmianie)
- Podczas kompilacji uruchamiane są testy jednostkowe
- Ogranicza ryzyko, że dostarczony system nie będzie w ogóle działać
- Pozwala na szybką identyfikację defektów
- Umożliwia prezentację systemu w każdej chwili
- Narzędzia wspierające CI to system kontroli wersji (np. Git) oraz serwer CI/CD (np. Jenkins)

Ciągłe dostarczanie (Continuous Delivery)

- Praktyka zakładająca, że projekt jest budowany w taki sposób, że może on zostać wdrożony na środowisko produkcyjne w każdej chwili
- Regularne dostarczanie oprogramowania w krótkich cyklach (CD jest ściśle powiązane z iteracyjnym wytwarzaniem oprogramowania)
- CI jest jednym z etapów CD
- Jeżeli projekt jest stabilny (kompilacja i testy jednostkowe zakończyły się powodzeniem), to projekt jest automatycznie umieszczany na środowisku testowym
- Kolejnym krokiem procesu jest uruchomienie testów automatycznych na gotowym systemie
- Często stosowaną praktyką są tzw. mechanizmy „feature-toggle" zakładające, że wdrażamy funkcjonalność, ale jest ona ukryta
- Wdrożenie na produkcję następuje poprzez ręczne wywołanie
- Rozwinięciem Continuous Delivery jest Continuous Deployment, gdzie cały proces od zbudowania aplikacji do jest wdrożenia odbywa się automatycznie

Metryki i raportowanie

Testy są monitorowane i mierzone w celu uzyskania informacji zwrotnej oraz umożliwienia wglądu w proces testowy.

- Stosunek pracy wykonanej przy m.in. przygotowaniu testów, zestawieniu środowiska i wykonania testów do całego kosztu testów
- Status wykonania przypadków testowych (ilość wykonanych/oczekujących, zaliczonych/niezaliczonych)
- Liczba odnalezionych i naprawionych/nienaprawionych defektów (dla całego projektu lub np. scenariusza)
- Metryki są nieodłącznym elementem raportu z testów, który zbiera je w całość

Ryzyka projektowe

- Obszary ryzyka związane ze zdolnością osiągnięcia celu projektu
- Czynniki organizacyjne – brak kompetencji, problemy kadrowe, słabo zdefiniowany proces, brak strategii
- Niedocenienie wartości testowania
- Niedostarczenie odpowiednich zasobów przez innych dostawców (np. systemy zewnętrzne)
- Niska jakość wymagań lub ich niekompletność oraz ich częste zmiany

Ryzyka produktowe

- Opisują potencjalne obszary i przyczyny wystąpienia defektów w oprogramowaniu
- Dostarczenie oprogramowania posiadającego defekty
- Niska jakość początkowa
- Możliwość wyrządzenia szkody człowiekowi lub organizacji przez wadliwy system
- Słabe parametry oprogramowania (użyteczność, bezpieczeństwo, wydajność, funkcjonalność)
- System nie spełnia wymagań (jest niewłaściwy)

Narzędzia wspierające testowanie

- Narzędzie do zarządzania testami – pozwalają tworzyć raporty, scenariusze i przypadki testowe, tworzyć i śledzić defekty, zbierać wymagania (np. Jira)
- Narzędzia do modelowania (przypadki użycia, diagramy sekwencji, aktywności, mapy myśli)
- Narzędzia do wykonywania testów – pozwalają tworzyć i uruchamiać testy automatyczne (np. Java + Maven + TestNG + Selenium WebDriver)
- Komparatory tekstu – pozwalają w łatwy sposób znaleźć różnice w dwóch plikach, np. różnice między oczekiwanym a rzeczywistym interfejsem mikroserwisu, np. WinMerge
- Dodatki do przeglądarki – tworzenie zrzutów ekranów, modyfikacja ciasteczek, monitorowanie ruchu sieciowego na stronie, modyfikacja źródła strony
- Interaktywne proxy – pozwalają monitorować i manipulować żądaniami wysyłanymi do aplikacji, np. Charles Web Debugging Proxy
- Narzędzia do testów API – pozwalają wysyłać żądania do usług sieciowych SOAP, REST, np. SoapUI, Postman
- Narzędzia do testów obciążeniowych – pozwalają symulować wielu użytkowników korzystających z aplikacji i mierzyć czas odpowiedzi, np. Jmeter
- Generatory danych testowych, np. numerów NRB, PESEL itp.
- Narzędzia ciągłej integracji i dostarczania – umożliwiają wersjonowanie, budowanie i udostępnianie aplikacji do testów, np. Git, Jenkins, Ansible

Przykładowe pytania z rozmów kwalifikacyjnych

- Jakie znasz rodzaje testów (wraz z ogólnym ich opisaniem)?
- Czym są błąd, defekt, awaria?
- Co powinien zawierać zgłoszony defekt?
- Co to są testy dymne i testy regresji?
- Czym różni się testowanie od zarządzania jakością?
- Czym jest weryfikacja, a czym walidacja?
- Czym są scenariusze i przypadki testowe?
- Jakie są cele testowania?
- Czym różni się wytwarzanie kaskadowe od iteracyjnego?
- Czym jest back-end i front-end?
- Z jakimi narzędziami miałeś/miałaś okazję pracować?
- Czym jest ciągła integracja i ciągłe dostarczanie?
- Przykładowe pytania z odpowiedziami: http://testpro.pl/30-pytan-rekrutacyjnych/

Polecane materiały

- Strategia i plan testów:
  https://www.softwaretestinghelp.com/difference-between-test-plan-test-strategy-test-case-test-script-test-scenario-and-test-condition/

  https://www.softwaretestinghelp.com/writing-test-strategy-document-template/

  https://www.softwaretestinghelp.com/test-plan-sample-softwaretesting-and-quality-assurance-templates/
- CI/CD (fajne wyjaśnienie, polecam):
  http://arch.astrotech.io/proces-wytwarzania-oprogramowania/procesy-ci-cd.html