Faza konstrukcji PhotoCHAD

Łukasz Klasiński Marcin Witkowski

8 stycznia 2019

Przykłady testów funkcjonalnych

1) Dodawanie plików graficznych, ustawianie tagów, synchronizacja oraz portale społecznościowe:

   - Tester dodaje pliki niebędące plikami graficznymi
   - Tester próbuje dodać do biblioteki bardzo duży zbiór zdjęć (> 100k)
   - Tester nadaje tagom nietrywialne nazwy (np. chińskie znaki, znaki puste, etc.)
   - Tester loguje się do serwisu społecznościowego, po czym na stronie serwisu wygasza sesję.

2) Dodawanie i wyszukiwanie ludzi po wizerunkach:

   - Tester próbuje dodać dwie osoby o takim samym imieniu i nazwisku
   - Tester w czasie działania programu usuwa wszystkie zdjęcia na których jest jakaś osoba
   - Tester dodaje zdjęcia, na których osoba ma na sobie maskę z wizerunkiem innej osoby

3) Wykrywanie duplikatów:

   - Tester dodaje kilkaset plików różniących się maksymalnie 10 pikselami
   - Tester dodaje parenaście zdjęć, których barwy zostały przesunięte w fazie
   - Tester dodaje zdjęcia z odwróconymi kolorami

Pomiary na podstawie norm ISO/IEC 9126 i 25 000 wymagań niefunkcjonalnych

1) Testowanie interfejsu: Przeprowadzamy testy obsługi interfejsu w różnych grupach wiekowych i na podstawie ich opinii oraz obserwacji stwierdzamy, czy wymaganie zostało spełnione.

2) Testowanie sieci neuronowej Przeprowadzamy wielokrotne testy na różnorodnych danych, sprawdzając czy sieć działa poniżej założonego progu błędu w każdych warunkach.

3) Testowanie wersji językowych Testujemy zmieniając wersje językowe i sprawdzając, czy nie ma żadnych błędów jak np. wymieszanie się słownictwa z dwóch różnych języków.

4) Testowanie zużycia zasobów stacjonarnych/mobilnych Uruchamiamy program i emulujemy typowe korzystanie z aplikacji analizując wpływ aplikacji na dostępne zasoby systemu. Testy są przeprowadzane na systemach o różnych specyfikacjach, aby przetestować zachowanie, kiedy zasobów jest za mało albo występuje ich nadmiar. Ponad to testujemy wpływ aplikacji na czas działania wszelakich urządzeń mobilnych.

Plan beta-testowania

1) Oddzielne grupy testerów dla

   - programistów
   - programistów nie związanych z projektem
   - osób trzecich

2) Zadania dla grup:

   - Programiści związani z projektem mają za zadanie używając wiedzy o systemie przetestować szczególnie takie dane, które mogły nie być wcześniej uwzględnione oraz bardziej 'wrażliwe' funkcjonalności.
   - Programiści spoza projektu zagrają rolę pentesterów próbujących popsuć aplikację na wszystkie możliwe sposoby
   - Osoby trzecie będą testować poprzez normalne użytkowanie aplikacji przez okres około miesiąca

3) Wszystkie przesłane zgłoszenia zostaną poddane analizie oraz poprawkom

4) Po wprowadzeniu poprawek, aktualizacja systemu oraz powtarzanie procesu do uzyskania pewności co to jakości oprogramowania

Plan zarządzania ryzykiem

Proces planowania oraz testowania powinien wyeliminować większość błędów, ale nigdy nie można mieć stu procentowej pewności, że nie zostaną odkryte nowe. Na wypadek wykrycia błędów, te będą przechodziły następującą procedurę:

1) Błąd zostaje zgłoszony przez użytkownika (bądź wykryty przez programistę)

2) Weryfikacja istnienia błędu oraz nadanie priorytetu

3) Zadanie trafia do odpowiedniego 2-tygodniowego sprintu na podstawie priorytetu (duży - jak najszybciej, mały kiedy nie ma nic innego do roboty)

4) Właściwe wprowadzenie poprawek oraz wgranie nowych wersji na odpowiednie platformy.
   Kolejnym ryzykiem jest popularność produktu. Jeśli baza użytkowników okaże się niezadowalająca, planujemy przygotowanie funduszy na przeprowadzenie kampanii reklamowej. Jeśli po przekroczeniu przyznanej ilości funduszy sytuacja się nie poprawi, będziemy zmuszeni przerwać projekt.

Plan zarządzania jakością oprogramowania

1) Regularne wykonywanie przygotowanych testów na nowych wersjach systemu.

2) Ankiety wśród użytkowników dotyczące takich aspektów jak design, zadowolenie z użytkowania, niezawodność itp.

3) W przypadku większych zmian, regularne monitorowanie oraz omawianie postępów w pracach.

4) Kontrolowanie działania systemu na innych platformach, szczególnie przy większych zmianach w systemach (np. nowa wersja androida, jak aplikacja radzi sobie w nowych rozdzielczościach itp.).

5) Przeprowadzanie statystyk wykresów, raportów dotyczących błędów, oraz czasów ich naprawiania, które pozwolą kontrolować stan jakości systemu. Ponadto przechowywanie ich w archiwum w celach przyszłościowej referencji (w trakcie pracy innych aplikacjach).

6) Przyznawanie odpowiednich specjalistów do teamów zajmujących się błędami z konkretnej dziedziny (np. specjalista od sieci neuronowych od błędów w picture recognition).

Plan wykonania produktu

Do zarządzania projektem oraz sprintami będziemy korzystali z narzędzia *Targetprocess*.

Podczas implementowania oprogramowania `PhotoCHAD` będziemy pracować zgodnie z metodyką Scrum. Zakładamy sprinty o długości dwóch tygodni. Każda z iteracji (sprintów) będzie składała się z:

- Implementacji wybranego zadania w ramach aktualnej iteracji
- Przetestowanie behawioralne zaimplementowanej funkcjonalności przez członka zespołu, który nie był z nią powiązany
- Sprawdzenie pokrycia unit-testami
- Zakończenie iteracji i merge do głównego projektu

Standardowo w czasie końcowym sprintu będą organizowane spotkania mające na celu zebranie opinii pracowników na temat tego jak oceniają poczęte zmiany. Na końcu iteracji odbywa się *Sprint Review*.

Ocena zgodności

W porównaniu z poprzednimi koncepcjami, nie zaszły duże zmiany. Została doprecyzowana metodyka wykonania produktu oraz dodany plan zarządzania jakością. Pozostałe elementy pozostały bez zmian. Dokładniejsza analiza zostanie przeprowadzona po wykonaniu testów, gdzie sprawdzona zostanie zgodność produktu z projektem.