Inżynieria Oprogramowania

Projekt cz. 3

Joanna Górecka, Aleksandra Broda

CONQUEST

1. Testy funkcjonalne

A. Firma produkująca czekoladę odkrywa czekoladowy świat stworzony przez konkurencyjną firmę. Przegląda ilość wyświetleń i komentarze, aby sprawdzić, jak dobry mają odbiór w aplikacji.

1) Test czy ilość wyświetleń jest prawidłowo wyświetlana.
2) Sprawdzenie, jak szybko ilość wyświetleń się aktualizuje.
3) Przetestowanie wyświetleń pod kątem nadużyć, aby zapobiec nadmiernemu nabijaniu wyświetleń za pomocą automatyzacji.
4) Sprawdzenie, czy prawidłowo działa moduł blokowania dodawania więcej niż x komentarzy na minutę.
5) Przetestowanie czy faktycznie obowiązuje limit x znaków na komentarz i czy akceptowane są komentarze krótsze niż 10 znaków.
6) Testowanie systemu komentarzy pod kątem użycia przez użytkownika niemile widzianych znaków specjalnych.
7) Testowanie systemu mającego za zadanie wykrycie zautomatyzowanego dodawania komentarzy przez boty.

B. Użytkownik chce dowiedzieć się, jak wysoko jest w rankingu globalnym. Otwiera go i sprawdza swoją pozycję w nim, aby móc pochwalić się tym znajomym.

1) Testowanie szybkości aktualizacji rankingu globalnego.
2) Testowanie jak radzi sobie ranking z nadmiernym obciążeniem ze strony użytkowników.
3) Testowanie opcji czasowej częściowej blokady aktualizacji pól, które w sposób niekontrolowany (bardzo szybki) tracą na aktualności.
4) Testowanie działania rankingu globalnego w trybie awaryjnym offline, czyli przy aktualizacji tylko raz na 10 minut.
5) Sprawdzenie czy użytkownikom przypisywane jest właściwe miejsce w rankingu.

C. Użytkownik aplikacji chce sprawdzić, jak wyglądają jego statystyki postaci po przejściu ostatniego świata. Wchodzi w informacje o awatarze i wyświetla swoje aktualne umiejętności i ich poziomy.

1) Test poprawnego wyświetlania statystyk poszczególnych użytkowników.
2) Test w momencie nieoczekiwanej aktualizacji statystyk z dwóch niedozwolonych sesji jednocześnie (użytkowanie jednego konta przez kilka osób).
3) Testowanie statystyk w momencie kiedy historia chce zmodyfikować je w sposób niedozwolony.
4) Sprawdzenie górnych limitów statystyk.

2. Pomiary dla wymagań niefunkcjonalnych

Wymagania niefunkcjonalne:

- Aplikacja powinna działać zarówno na urządzeniach mobilnych z Androidem oraz z iOS
    - **Pomiar:** X = A/N, Y = B/M
    - N - liczba funkcji programu, M - liczba niezbędnych funkcji programu
    - A - liczba funkcji działająca na obu systemach, B liczba niezbędnych funkcji działająca na obu systemach
    - **Cel:** X >= 0.99, Y = 1.00

- Interfejs powinien być czytelny i intuicyjny
    - **Pomiar:** X = A/T
    - A - liczba przypadków, w których użytkownik miał problem z obsługą interfejsu (nieczytelność, niejasne przeznaczenie, niezrozumiałe instrukcje, dłuższe zastanawianie się nad tym co dalej zrobić, itd.)
    - T - czas użytkowania w minutach
    - **Cel:** X <= 0.005

- Błędy aplikacji i/lub połączenia nie powinny zupełnie wymazywać postępów użytkownika
    - **Pomiar:** X = A/N
    - N - liczba testów, w których umyślnie zostały wywołane błędy aplikacji/połączenia
    - A - liczba przypadków, w których postępy użytkownika sprzed ostatnich 3 minut przerwanej sesji nie zostały zapisane.
    - **Cel:** X <= 0.005

- Każdy użytkownik powinien mieć przypisany unikalny kod identyfikacyjny
    - **Pomiar:** X = A/N
    - N - liczba wywołań funkcji przydzielającej kod identyfikacyjny w trakcie testu
    - A - liczba przydzielonych w trakcie testu kodów, które są unikatowe
    - **Cel:** X = 1.00

- Aplikacja powinna być dostępna w języku polskim i angielskim
    - **Pomiar:** X = A/N
    - N - liczba elementów interfejsu korzystająca z języka pisanego lub mówionego (nie dotyczy przygód tworzonych przez użytkowników aplikacji)
    - A - liczba tych elementów dostępna zarówno w języku polskim, jak i angielskim
    - **Cel:** X = 1.00

- Aplikacja nie powinna zajmować zbyt dużej ilości pamięci dyskowej
    - **Pomiar:** X = A/M
    - N - maksymalny oczekiwany rozmiar wersji klienckiej aplikacji w pamięci dyskowej w momencie instalacji (100 MB)
    - A - maximum z otrzymanych rozmiarów różnych wersji klienckich aplikacji w momencie instalacji (podane w MB)
    - **Cel:** X <= 1.00

- Aplikacja powinna korzystać z bezpiecznego połączenia (zapewniać bezpieczeństwo danych)
    - **Pomiar:** X = A/N
    - N - liczba wymienionych danych wymagających ochrony
    - A - liczba wymienionych i podlegających ochronie danych, która została zaszyfrowana
    - **Cel:** X = 1.00

- Ładowanie przygody nie powinno trwać za długo
    - **Pomiar:** X = T/M
    - M - maksymalny oczekiwany czas ładowania przygody (30 s)
    - T - maximum z czasów ładowania osiągniętych w trakcie testów (podane w sekundach)
    - **Cel:** X <= 1.00

- Ładowanie rankingów nie powinno trwać za długo
    - **Pomiar:** X = T/M
    - M - maksymalny oczekiwany czas ładowania rankingu (8 s)
    - T - maximum z czasów ładowania osiągniętych w trakcie testów (podane w sekundach)
    - **Cel:** X <= 1.00

3. Plan beta-testowania

Beta-testowanie zostanie podzielone na trzy fazy:

Faza I (dwa miesiące)
Aplikacja będzie testowana przez dwóch betatesterów, którzy na cały etat będą sprawdzać aplikację pod kątem błędów.

Sprawdzą najważniejsze funkcjonalności aplikacji, a w szczególności:

- System logowania/kont użytkowników pod kątem bezpieczeństwa, czy niepowołani użytkownicy nie mają dostępu do wrażliwych danych;
- Funkcjonalności aplikacji, system dodawania i przechodzenia światów, jak naliczane są punkty statystyk, poprawność wyświetlania użytkowników w rankingach, opcje pomocnicze takie jak dodawanie komentarzy i ocen;

- Szczególną uwagę trzeba poświęcić systemowi losowania unikalności każdego przechodzenia historii, aby mieć pewność, że użytkownicy są traktowani "sprawiedliwie".

Faza II (po zakończeniu większości Fazy I)
Aplikację testować będzie cały zespół programistyczny oraz osoby zaprzyjaźnione z projektem, które uzyskały akceptację kierownika projektu (w ramach wolontariatu).
W tym wypadku przeprowadzone będą wyrywkowe testy aplikacji, na zasadzie poszukiwania wszelkich nieprawidłowości w systemie, na podstawie indywidualnych zainteresowań poszczególnymi elementami aplikacji.

Faza III (testowanie przy udziale realnych użytkowników aplikacji)
Na koniec zostanie otwarta możliwość testowania aplikacji przez końcowych użytkowników aplikacji.
Zostanie wprowadzony program wyłapywania wszelkich niedociągnięć - osoby, zgłaszające najwięcej krytycznych błędów otrzymają dodatkowe przywileje w grze np. możliwość darmowego stworzenia komercyjnego świata lub szybszy dostęp do niektórych osiągnięć, czy specjalne statystyki postaci na start.
Najaktywniejsi wolontariusze biorący udział w testach zostaną wpisani na listę *hall of fame* dostępną w aplikacji.

Ważnym etapem tej fazy jest sprawdzenie jak aplikacja poradzi sobie pod obciążeniem - równoczesne dodawanie Światów, przechodzenie ich przez różnych użytkowników, bieżąca aktualizacja rankingów.

4. Plan zarządzania ryzykiem

Skala prawdopodobieństwa: 1-10
Możliwe rangi ryzyka: Marginalne / Mało ważne / Ważne / Krytyczne / Katastroficzne

1) **Źródło ryzyka:** Spam (np. publikowanie tej samej historii pod wieloma tytułami)
   Prawdopodobieństwo wystąpienia: 10
   Ranga: Mało ważne
   Sposób rozwiązania problemu: Wyznaczenie moderatorów, system zgłaszania spamu, weryfikacja skarg przez moderatorów, usuwanie spamu, banowanie spamujących użytkowników.

2) **Źródło ryzyka:** Niepoprawnie stworzone przygody (pusta zawartość, historie stworzone tylko z jednego slajdu, brak tagów, opisu, itd.)
   Prawdopodobieństwo wystąpienia: 10
   Ranga: Marginalne
   Sposób rozwiązania problemu: Proces publikacji przygody wymaga od twórcy wypełnienia tych danych oraz spełnienia przez historię określonych kryterium dotyczących długości, powtarzania się slajdów, itd.

3) **Źródło ryzyka:** Nieodpowiednie treści w tworzonych przygodach (promowanie rasizmu, znęcanie się nad zwierzętami, itp.)
   Prawdopodobieństwo wystąpienia: 9.5
   Ranga: Ważne
   Sposób rozwiązania problemu: System zgłaszania historii, weryfikacja skarg przez moderatorów, usuwanie tych historii, banowanie użytkowników.

4) **Źródło ryzyka:** Plagiaty
   Prawdopodobieństwo wystąpienia: 10
   Ranga: Ważne
   Sposób rozwiązania problemu: System zgłaszania plagiatów, weryfikacja skarg przez moderatorów, usuwanie tych historii, banowanie użytkowników. Jeśli to konieczne, zgłoszenie incydentu odpowiednim służbom.

5) **Źródło ryzyka:** Wyzyskiwanie systemu przez użytkowników (np. tworzenie bardzo krótkich historii, które nagradzają graczy olbrzymią ilością punktów czy też hakowanie aplikacji)
   Prawdopodobieństwo wystąpienia: 10
   Ranga: Krytyczne
   Sposób rozwiązania problemu: Ograniczenia na ilość nagradzanych punktów w stosunku do długości historii, wymaganego poziomu. Narzędzia analityczne mające na celu wykrycie podejrzanych czynności.

6) **Źródło ryzyka:** Przeciążenie serwerów
   Prawdopodobieństwo wystąpienia: 7
   Ranga: Krytyczne
   Sposób rozwiązania problemu: Analiza i przewidywanie ruchu. Przygotowanie serwerów na zwiększony ruch w czasie trwania promocji. Ewentualnie dołączenie kolejnych serwerów wraz ze wzrostem zainteresowania aplikacją.

7) **Źródło ryzyka:** Próby kradzieży danych użytkowników
   Prawdopodobieństwo wystąpienia: 9
   Ranga: Krytyczne
   Sposób rozwiązania problemu: Zatrudnienie specjalistów od bezpieczeństwa. Korzystanie z protokołów typu SSH i SSL/TLS. Szyfrowanie przechowywanych danych. Częste aktualizowanie oprogramowania zapewniającego bezpieczeństwo.

8) **Źródło ryzyka:** Złe określenie harmonogramu wykonania, opóźnienia
   Prawdopodobieństwo wystąpienia: 5
   Ranga: Katastroficzne
   Sposób rozwiązania problemu: Reewalucja stanu prac i planu wykonania, stworzenie nowego harmonogramu, zatrudnienie nowych pracowników, rezygnacja z dodatkowych, zbędnych funkcjonalności, w najgorszym przypadku przedłużenie czasu projektu i/lub zwiększenie budżetu projektu.

9) **Źródło ryzyka:** Problemy w wykonaniu spowodowane przez dezinformację i brak komunikacji
Prawdopodobieństwo wystąpienia: 9
Ranga: Krytyczne
Sposób rozwiązania problemu: Podzielenie pracowników na mniejsze podzespoły, codzienne spotkania w zespołach oraz między koordynatorami zespołów w celu synchronizacji wykonywanych prac i promowania komunikacji między pracownikami. Przygotowanie odpowiednich szkoleń dla pracowników

10) **Źródło ryzyka:** Niewystarczający budżet
Prawdopodobieństwo wystąpienia: 6
Ranga: Katastroficzne
Sposób rozwiązania problemu: Zmniejszenie ilości pracowników, wybranie tańszych lub darmowych wersji oprogramowania, rezygnacja z dodatkowych, zbędnych funkcjonalności, w najgorszym przypadku zwiększenie budżetu projektu.

11) **Źródło ryzyka:** Brak zainteresowania aplikacją
Prawdopodobieństwo wystąpienia: 5
Ranga: Krytyczne
Sposób rozwiązania problemu: Zatrudnienie specjalistów od reklamy. Organizacja konkursów, promocji, itp.

12) **Źródło ryzyka:** Problemy w realizacji logiki projektu
Prawdopodobieństwo wystąpienia: 4
Ranga: Katastroficzne
Sposób rozwiązania problemu: Jeśli to możliwe wyodrębnienie problematycznych części, redakcja planu wykonania, rezygnacja z części funkcjonalności, zmiana wykorzystywanych technologii, w najgorszym przypadku zwiększenie budżetu i/lub czasu projektu.

13) **Źródło ryzyka:** Wystąpienie nieoczekiwanych błędów w oprogramowaniu (po wypuszczeniu oprogramowania na rynek)
Prawdopodobieństwo wystąpienia: 8
Ranga: Krytyczne
Sposób rozwiązania problemu: System zgłaszania problemów przez użytkowników, weryfikacja problemu przez testerów, możliwie szybkie naprawienie błędu, sprawdzenie poprawności nowej wersji i opublikowanie aktualizacji z poprawką.

5. Plan zarządzania jakością wytwarzania oprogramowania

1) Testowanie aplikacji przez cały okres życia oprogramowania przez różne podmioty.

2) Opracowanie statystyk i wyników testów dostępnych dla uprawnionych uczestników projektu.

3) Szczegółowa analiza przeprowadzonych testów, a po niej dodatkowe testowanie najsłabszych ogniw, które zostały znalezione w czasie analizy.

4) Kierownik projektu kontroluje na bieżąco czy efekty pracy poszczególnych członków zespołu pokrywają wszelkie postawione przed zespołem wymagania.

5) Audyt oprogramowania przez firmę zewnętrzną po ukończeniu 80% prac nad projektem.

6. Dokładniejszy planu wykonania produktu

W wytwarzaniu naszego oprogramowania będziemy korzystać z *realizacji przyrostowej*.

Harmonogram wykonania:

1) Określenie wymagań (1 miesiąc)
2) Ogólny projekt aplikacji (3 tygodnie)
3) Iteracje: (6 miesięcy)
    a) Wybór podzbioru funkcji
    b) Szczegółowy projekt
    c) Implementacja
    d) Testy
    e) Dostarczenie zrealizowanej części systemu
4) Faza I beta-testów i poprawki (1 miesiąc, start 2 tygodnie przed końcem iteracji)
5) Start reklamowania produktu (tydzień przed zakończeniem iteracji)
6) Faza II beta-testów, poprawki oraz szkolenie moderatorów, adminów (2 tygodnie, start po zakończeniu iteracji),
7) Faza III beta-testów i ostatnie poprawki (3 tygodnie)
8) Przygotowania do udostępnienia aplikacji (1.5 tygodnia)
9) Publikacja oprogramowania
10) Utrzymanie oprogramowania (nieokreślone)

Ocena pracochłonności: 9.5 miesiąca

7. Ocena zgodności wykonanych prac z wizją systemu i specyfikacją wymagań

Zgodność wykonywanych prac z wizją systemu i specyfikacją wymagań uważamy za w głównej mierze zachowaną. Główną zmianą jest

wydłużenie czasu projektu. Prawdopodobnie zajdzie również potrzeba powiększenia zespołu.

Dopiero zaprojektowanie konkretnych testów uświadomiło nam złożoność elementów wpływających na komfort użytkowania naszej aplikacji. Obecne oszacowania to wciąż nie jest kwestia do końca rozstrzygnięta. Jeżeli testy wykażą nieprawidłowości w działaniu elementów - np. ranking punktów po dokładnych testach przy dużym ruchu użytkowników będzie działał zbyt wolno, to prace nad aplikacją mogą nieoczekiwanie się przedłużyć. Ciężko powiedzieć, czy czas przeznaczony na poprawki aplikacji po przetestowaniu będzie wystarczający, aby wypuścić na rynek gotowy produkt. Niedociągnięcia bardzo szybko mogą skutkować klęską - brakiem zainteresowania użytkowników naszą grą.