Politechnika Łódzka
Wydział Elektrotechniki, Elektroniki,
Informatyki i Automatyki

Praca dyplomowa magisterska

Testowalność aplikacji mobilnych na platformę Android

Rafał Sowiak
Nr albumu: 199564

Opiekun pracy: prof. dr hab. inż. Andrzej Napieralski
Dodatkowy opiekun: prof. mgr inż. Michał Włodarczyk

Łódź, 26.02.2016

Spis treści

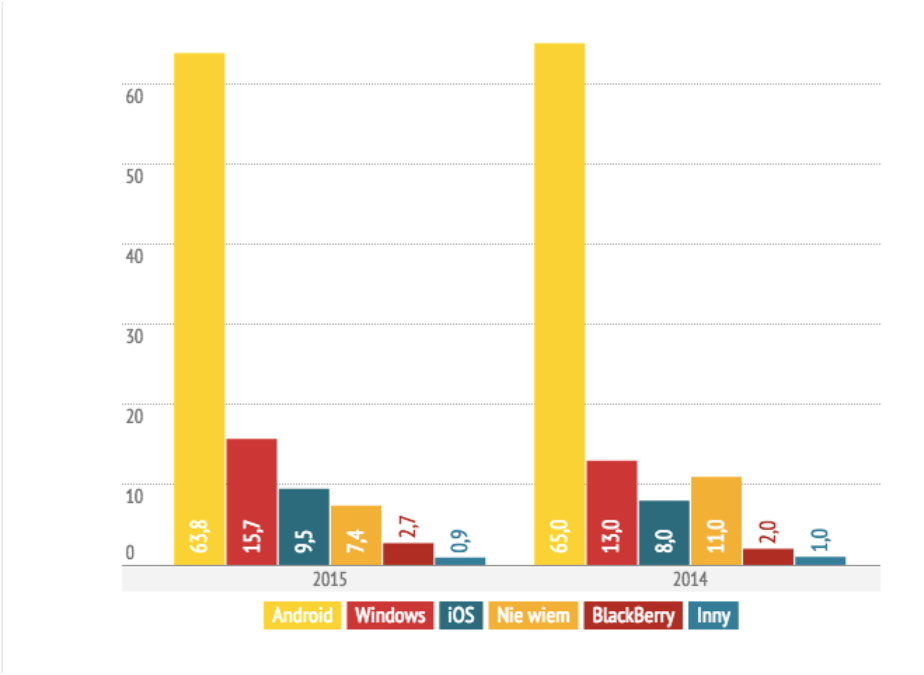
Rozdział 1

Wprowadzenie

Rozdział 2

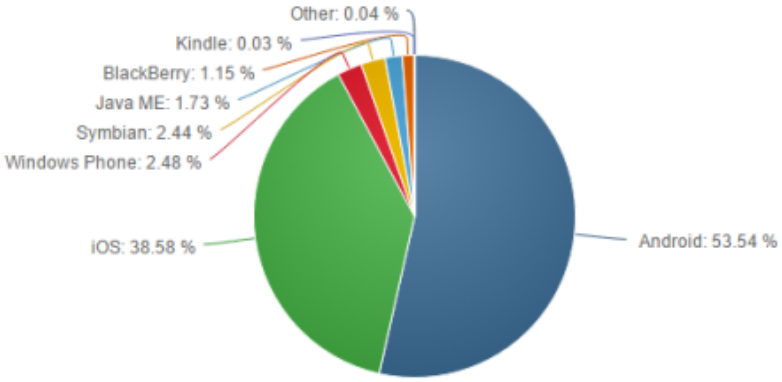
Android jako system operacyjny

Android – według wiadomości zawartych w Wikipedii – to system operacyjny z jądrem Linux, przeznaczony dla urządzeń mobilnych, takich jak telefony komórkowe, smartfony, tablety (tablety PC) i netbooki. Według portalu AntyWeb.pl w maju 2015 roku system ten miał największy udział w rynku urządzeń mobilnych w Polsce, a wartość tego udziału od 2014 roku to około 65



Rysunek 2.1: Android – udział w rynku urządzeń mobilnych w Polsce. (Źródło: portal AntyWeb.pl, 05/2015)

Drugie miejsce, według tego samego portalu, zajmuje system Windows, a trzecie iOS. Na świecie proporcje udziału są nieco inne, ale ciągle na czele jest system Android, z wynikiem ponad 53%, natomiast tutaj na drugim miejscu plasuje się już wyraźnie iOS z niemal 40-procentowym udziałem w rynku.



Rysunek 2.2: Android – udział w rynku urządzeń mobilnych na świecie. (Źródło: portal android.com.pl, 09/2015)

Jako system operacyjny dostępny nieodpłatnie, Android zrzesza przy sobie dużą społeczność deweloperów piszących aplikacje, które poszerzają funkcjonalność urządzeń. W sierpniu 2014 roku w Google Play (wcześniej Android Market), dostępnych było ponad 1,3 miliona aplikacji, zarówno komercyjnych, jak i darmowych.

Najpopularniejszymi językami programowania, w których piszemy aplikacje na Androida, są Java i C++ ze środowiskiem Android NDK. O ile język Java wydawać się może najrozsądniejszym wyborem na pierwszy rzut oka, o tyle wielu programistów używa również środowiska NDK. Jest to zestaw narzędzi, który pozwala realizować części swojej aplikacji za pomocą kodu macierzystego języków takich jak C i C ++. Zazwyczaj środowisko to wykorzystuje się w celu pisania programów, które intensywnie wykorzystują CPU, takich jak silniki gier, przetwarzania sygnału i symulacji fizyki. Jednak deweloperzy muszą wziąć pod uwagę również wady takiego rozwiązania, które mogą nie do końca zbilansować korzyści. Natywny kod Androida na ogół nie powoduje zauważalnej poprawy wydajności, ale za to zawsze znacznie zwiększa złożoność aplikacji, który to problem i tak już jest dużym wyzwaniem dla programistów Java, co przybliżę w kolejnych rozdziałach. Podsumowując, z NDK należy korzystać tylko wtedy, jeśli uznamy, że jest to niezbędne dla wytwarzanej aplikacji, a nie dlatego, że lepiej czujemy się programując w języku C/C++. Zanim zdecydujemy się na to rozwiązanie, najpierw należy sprawdzić, czy androidowe API zapewnia funkcjonalność, której potrzebujemy.

Java i C czy C++ to jednak nie wszystkie języki, których możemy użyć przy programowaniu aplikacji na Androida. Podczas szukania materiałów do tej pracy spotkałem się z przykładami aplikacji napisanych w C#, Delphi, czy nawet PHP. Stanowią one tak mały procent, że postanowiłem nie brać ich pod uwagę analizując opisywany problem testowalności oprogramowania na ten system operacyjny.

Rozdział 3

Testowalność oprogramowania

Jako analityk testowy z pojęciem testowalności oprogramowania spotykam się od początku mojej pracy w branży. Termin „testowalność oprogramowania" [1] według definicji ISTQB [2] i ISO9126, to najkrócej mówiąc właściwość tego oprogramowania umożliwiająca testowanie go po zmianach. Termin ten ściśle powiązany jest także z innym pojęciem ze słownika testerskiego – pielęgnowalnością. A pielęgnowalność [3] , to łatwość, z którą oprogramowanie może być modyfikowane w celu naprawy defektów, dostosowania do nowych wymagań, modyfikowane w celu ułatwienia przyszłego utrzymania lub dostosowania do zmian zachodzących w jego środowisku.

Po co tak w ogóle właściwie testować oprogramowanie? Czy testowanie jest potrzebne? Jak dużo testów należy przeprowadzić, aby testowanie było wystarczająco skuteczne? To są pytania, które mogą być tematem osobnej pracy, ale ja w skrócie postaram się na nie odpowiedzieć w tym rozdziale.

Człowiek, jako istota żywa i omylna, może popełnić podczas pracy **błąd**, czyli inaczej – **pomyłkę**. Pomyłka w pracy programisty może skutkować pojawieniem się **defektu** (usterki, pluskwy) w kodzie programu, bądź w dokumencie. Do tej pory nic się nie dzieje złego, ale jeżeli kod programu, który posiada w sobie taki defekt, zostanie wykonany, system może nie zrobić tego, co od niego wymagamy, lub wykonać to niezgodnie z założeniami, czyli inaczej rzecz ujmując, ulegnie **awarii**.

Defekty powstają, ponieważ jak już wspomniałem wcześniej, ludzie są omylni. Ale pomyłka człowieka, to nie jedyny powód awarii systemów. Mogą one być również spowodowane przez warunki środowiskowe, takie jak promieniowanie, pole magnetyczne i elektryczne, czy nawet zanieczyszczenia środowiska.

Czy testowanie pomaga nam w całkowitym uniknięciu awarii systemów? Na pewno nie, ale pozwala je drastycznie ograniczyć. Za pomocą zestawu testów możemy zmierzyć jakość oprogramowania wyrażoną przez ilość znalezionych usterek oraz możemy budować zaufanie do jakości oprogramowania, jeżeli jako osoby testujące znajdujemy ich mało, bądź nie znajdujemy ich wcale.

I tutaj najważniejsze zdanie: Testowanie samo w sobie nie poprawia jakości oprogramowania! Dopiero znalezienie, gdzie defekt znajduje się w kodzie programu (zdebugowanie) oraz

[1] Definicja testowalności według standardu ISO9126

[2] International Software Qualification Board

[3] Definicja pielęgnowalności według ISTQB

Tabela 3.1: Koszty znalezienia błędu na poszczególnych etapach projektu

Błąd znaleziony podczas	**Szacowany koszt**
Projektowania	1 PLN
Inspekcji (przeglądu)	10 PLN
W początkowej fazie produkcji	100 PLN
Podczas testów systemowych	1000 PLN
Po dostarczeniu produktu na rynek	10000 PLN
Kiedy produkt musi zostać wycofany z rynku	100000 PLN
Kiedy produkt musi zostać wycofany z rynku po wyroku sądowym	1000000 PLN

naprawa tego błędu przez programistę, poprawi nam tą jakość. Poniżej na rysunku widzimy tabelę (słynną już, gdyż korzysta z niej wielu trenerów prowadzących kursy z testowania oprogramowania), jak testowanie na poszczególnych etapach wytwarzania oprogramowania wpływa na koszty projektu.

Z zestawienia jasno wynika, że praca testerów nie zaczyna się gdy program już jest napisany przez programistów, a zaczyna się już w najwcześniejszej fazie projektu, na etapie projektowania.

Jak to jest więc z tą testowalnością oprogramowania dla Android? Czy aktualna struktura stosowana w większości aplikacji jest testowalna? Czy, o ile nie jest, da się tak poprawić strukturę programów, aby były bardziej testowalne niż obecnie? Jak nie psuć wcześniej działających części aplikacji wprowadzanymi zmianami w kodzie? Jak w ogóle wykrywać takie sytuacje, gdy coś przypadkowo zepsuliśmy? Jak dzielić odpowiedzialność pomiędzy częściami naszego oprogramowania? Jak rozwiązać problem zbyt dużego sprzężenia zarówno w naszym kodzie, jak i pomiędzy naszym kodem i frameworkiem androidowym [4]? No i w ogóle, jak testować aplikacje dla Androida poprawnie?

Na te pytania postaram się odpowiedzieć w drugiej części mojej pracy.

[4]Sprzężenie (*ang. coupling*) jest miarą jak bardzo obiekty, podsystemy lub systemy zależą od siebie nawzajem.

Rozdział 4

Wnioski

Bibliografia

[1] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749.

Spis rysunków

Spis tabel