



Magazine

Podejścia anty-regresyjne: Analiza wpływu i porównawcze oraz mieszane testowanie regresyjne

Część I: Wprowadzenie i analiza wpływu

Autor: Paul Gerrard

O autorze: Paul jest konsultantem, nauczycielem, autorem, projektantem stron, programistą, testerem, prelegentem na konferencjach, coachem i wydawcą. Podejmował się usług konsultingowych we wszystkich aspektach testowania oprogramowania i zapewnienia jakości, specjalizując się w zapewnieniu testów. Prezentował wykłady i przewodniki na konferencjach w całej Europie, USA, Australii i Południowej Afryka i od czasu do czasu zdobywał za nie nagrody.

Wykształcony na uniwersytetach w Oxford i Imperial College London, Paul był założycielem Schematu Certyfikacji Testerów BCS ISEB oraz członkiem Grupy roboczej opracowującej BS 7925 – Standard dla testowania komponentów. Obecnie jest dyrektorem Gerrard Consulting Limited, dyrektorem Aqastra Limited i gospodarzem Forum Zarządzania Testowaniem w Wielkiej Brytanii.



Wprowadzenie

Przez kilka lat unikałem zbyt dużego zaangażowania w wykonywanie automatyzacji testów, ponieważ czułem, że to nudne. Tak, wiem, że to ma duże szanse i w zasadzie z pewnością powinniśmy przekazywać nudne, powtarzające się zadanie pisania testów regresji narzędziom. Ale myślę, że zasady testowania regresji nie zmieniły się i zrobiliśmy niewielki postęp przez, mniej więcej, ostatnich piętnaście lat - tak naprawdę, to nie ruszyliśmy się z miejsca. Zatrzymało się to w czasie.

Na Eurostar 1997 roku przedstawiłem referat na temat automatyzacji testów. Zatytułowany "Testowanie GUI aplikacji" – artykuł, który napisałem jest nadal najbardziej popularny na mojej stronie gerrardconsulting.com z liczbą pobrań około 300 na miesiąc. Dlaczego ludzie nadal są zainteresowani rzeczami, które napisałem tak dawno temu? Myślę, że to był dobry, ale nie przełomowy artykuł; nie wspominał o sieci; zalecenia dla automatyzacji testów były rozsądne, ale nie radykalne. Napisano już książki na ten temat. Zamierzałem zaktualizować ten dokument dla nowego, „połączonego” świata, jaki teraz mamy, ale nie miałem do tej pory czasu.

Ale w styczniu 2010 na konferencji *Test Management Summit*, wybrałem się na sesję "Testowanie regresyjne: Co automatyzować i jak". Podczas naszych przygotowań do konferencji wypłynął temat badania popularności. Musieliśmy włączyć go do programu, ale nikt nie zgłosił się na ochotnika – więc ja podjąłem się tematu. W tym dniu wyrzuciłem frustracje, które dręczyły mnie od dłuższego czasu i przemówienie stało się raczej złośliwą tyradą. W tym krótkim artykule chciałbym przedstawić przemyślenia zaprezentowane na konferencji.

Mam zamiar ponownie prześledzić nasze pierwsze kroki w automatyzacji i spróbować dowiedzieć się, dlaczego testerzy wciąż jeszcze odkrywają, że budowa i uruchomienie trwałego, znaczącego zestawu zautomatyzowanych testów regresji stwarza duże trudności. Może te trudności pojawiają się, bo nie myślimy o nich na początku?

Testy regresji są najbardziej stabilne i uruchamiane wielokrotnie, więc automatyzacja obiecuje duże oszczędności czasu a testy – zautomatyzowane – gwarantują rzetelne wykonanie i sprawdzanie wyników. Wybór automatyzacji wydaje się jasny. Ale chwileczkę!

Wykonujemy testy regresji, ponieważ zachodzą zmiany. Chaotyczne, niestabilne środowiska najbardziej potrzebują testów regresji. Ale kiedy rzeczy zmieniają się regularnie, automatyzacja jest bardzo trudna. I to opisuje jeden z paradoksów testów regresji. Środowiska programistyczne, które potrzebują i które skorzystałyby najbardziej ze zautomatyzowanych testów regresji są środowiskami, w których są największe trudności w implementacji tychże.

Przemyślenie regresji

Czym jest proces testowania regresji? W niniejszej pracy chcę, aby prześledzić przemyślenia związane z testami regresji. Aby zrozumieć, dlaczego regresja występuje; aby ustalić, co rozumiemy przez testy regresji, dlaczego decydujemy się je robić i je automatyzować.

Czym charakteryzuje się regresja?

Najkrócej - coś się zmienia i to wpływa na działające oprogramowanie. Może to być środowisko, w którym działa oprogramowanie, zaimplementowane ulepszenie lub poprawka jakiegoś błędu (i zmiana powodująca skutki uboczne) i tak dalej. Przez wiele lat mówiło się, że poprawki kodu oprogramowania dają 50% ryzyka wprowadzenia niepożądanych efektów w działającym oprogramowaniu. Czy 30% lub 80%? Kogo to obchodzi? Zmiana jest niebezpieczna; prawdopodobieństwo katastrofy jest trudne do przewidzenia; wszyscy tego doświadczyliśmy przez lata.

Regresje mają nieproporcjonalnie duży wpływ na ponowne prace, zaufanie, a nawet morale. Co możemy zrobić? Mamy do dyspozycji dwa podejścia – analiza oddziaływania (aby wspierać rozsądny wybór projektu), aby zapobiec regresji oraz testowanie regresji – aby zidentyfikować regresje, kiedy one wystąpią.

Analiza wpływu

W ocenie potencjalnych szkód, które może spowodować zmiana, oczywistym wyborem jest nie zmieniać w ogóle niczego. To stwierdzenie nie jest takie głupie, jak się wydaje. Od czasu do czasu, wartość dokonywania zmian, poprawy błędu, dodania cechy jest przeważona przez ryzyko wprowadzenia nowych, nieprzewidywalnych problemów. Wszystkie potencjalne zmiany muszą być ocenione pod kątem ich możliwego wpływu na istniejący kod i prawdopodobieństwa wprowadzenia niepożądanych efektów ubocznych. Problemem jest to, że ta ocena wpływu zmian - Analiza wpływu - jest niezwykle trudna.

Istnieją dwa punkty widzenia analizy wpływu: biznesowy i techniczny.

Ocena wpływu: Biznesowy punkt widzenia

Pierwszym z nich jest użytkownik biznesowego punktu widzenia: potencjalne zmiany są sprawdzane pod kątem tego, czy będą one miały taki wpływ na działanie systemu, jaki użytkownik może rozpoznać i zaakceptować. Popularne są trzy rodzaje wpływu na funkcjonalność: funkcjonalność wpływająca na biznes, dane lub proces.

Wpływy biznesowe mogą często powodować subtelne zmiany w zachowaniu się systemów. Przykładem może być sytuacja, gdy zmiana wpływa na to, jak interpretowana jest część danych: cena aktywów może być obliczana dynamicznie, a nie być stała przez cały okres użytkowania aktywów. Aktywa przechowywane w lokalizacji po jednej cenie, mogą być przeniesione w inne miejsce z inną ceną. Nagle - wartość nieistniejących aktywów w pierwszej lokalizacji jest pozytywna, a nawet ujemna! Jak to możliwe? Oprogramowanie działało idealnie - ale wpływ biznesowy nie był przemyślany.

Typowy wpływ na dane mógłby mieć miejsce, gdy element danych wymagany do zakończenia transakcji jest obowiązkowy, a nie opcjonalny. Możliwe, że obecni użytkownicy opierają się na opcjonalnych elementach danych, ponieważ w momencie wykonania danej transakcji, informacja nie jest znana, ale zdobywana później. "Ulepszony" system może zastopować wszystkie transakcje biznesowe lub zmusić użytkowników do wymyślania danych, aby obejść sprawdzanie poprawności danych. Tak czy inaczej, wpływ jest negatywny.

Funkcjonalność wpływająca na proces ma miejsce wtedy, gdy zmiana może mieć wpływ na wybór ścieżek w systemie lub na sam proces. Zmiana może na przykład spowodować wywołanie jakiejś funkcji systemu, podczas gdy wcześniej tak nie było. Alternatywnie, zmiana może ograniczyć korzystanie z funkcji, które użytkownicy już znali. Użytkownicy mogą odkryć, że muszą wykonywać niepotrzebną pracę lub utracili możliwość wykonywania pewnych istotnych zmian dotyczących transakcji. Tak nie może być!

Ocena wpływu: punkt widzenia techniczny

Odnosząc się do technicznej analizy wpływu wykonywanej przez projektantów lub programistów - istnieje szereg możliwości, a w niektórych środowiskach technicznych, odpowiednich narzędzi, które mogą pomóc. Mówiąc szerzej, ocena oddziaływania odbywa się na dwóch poziomach: z góry na dół (ang. *top-down*) i z dołu do góry (ang. *bottom-up*).

Analiza *top-down* wymaga rozważenia alternatywnych możliwości projektowania i przeanalizowanie ich wpływu na całościowe zachowanie zmienionego systemu. Aby naprawić błąd, zwiększyć funkcjonalność lub spełnić pewne nowe lub zmienione wymagania, mogą istnieć alternatywne projekty, które pozwolą osiągnąć te cele. Podejście *top-down* patrzy na te potencjalne zmiany w kontekście architektury jako całości, zasad projektowania oraz praktycznych aspektów wprowadzania samych zmian. Podejście to wymaga, by projektanci lub programiści mieli architektoniczny punkt widzenia, ale również powinien istnieć zbiór zasad projektowania lub wytycznych, które kierują ich z dala od złych praktyk. Niestety, niewiele organizacji ma taki punkt widzenia lub zasady projektowania na tyle mocno osadzone w ich zespołach, by mogli na nich polegać.

Analiza *bottom-up* jest oparta na kodzie. Jeśli wybrane podejście projektowania wpływa na zbiór komponentów, które się zmieniają, oprogramowanie, które wywołuje i zależy od komponentów „do zmiany” może być monitorowane (śledzone). Usługi wyższego poziomu i funkcje, które ostatecznie zależą od zmian, mogą być zidentyfikowane i ocenione. W zasadzie brzmi to dobrze, zwłaszcza jeśli istnieją narzędzia do tworzenia drzew połączeń i diagramów współpracy z kodu. Ale tu z kolei są dwa problemy.

Pierwszy problem jest taki, że integralność konstrukcji całego systemu może być niska. Zależności między zmienionymi komponentami i tymi, na które one mają wpływ mogą być liczne. Jeśli kod jest źle zorganizowany, zawiły i wygląda jak "spaghetti", nawet eksperci oprogramowanie mogą nie być w stanie pojąć tej złożoności i może się wydawać, że każda część systemu jest poddana wpływowi zmian. To jest przerażająca perspektywa.

Drugi problem to sytuacja, w której zmiany w oprogramowaniu mogą być na tak niskim poziomie hierarchii wywoływanych komponentów, że niepraktycznym jest śledzenie wpływu tych zmian poprzez funkcje wyższego poziomu. Pomimo, że zmieniany komponent może być osadzony dość głęboko w architekturze, efekty niepoprawnie zaimplementowanego oprogramowania mogą być katastrofalne. Dość wiadomą kwestią jest to, że wysokopoziomowe serwisy zależą od niskopoziomowych komponentów – problem w tym, że trudno jest ocenić te zależności, a tym samym oszacować wpływ proponowanych zmian.

Podsumowując, analiza wpływu to niełatwa perspektywa. Czy testy regresji mogą wyrwać nas z tej dziury?

Ciąg dalszy nastąpi...