Testowanie zgodnie z metodyką Agile – fakty i mity

Autor: Edward J. Correia

Firmy, które opracowują oprogramowanie zgodnie z metodyką Agile, czyli praktykują tzw. programowanie elastyczne (agile development), musiały prawdopodobnie stawić czoła takim samym wyzwaniom, jak George Wilson. Pełnił on funkcję szefa działu w firmie AIG Computer Services i jednocześnie zarządzał projektami związanymi z tworzeniem oprogramowania na komputery klasy średniej firmy IBM oraz komputery PC w środowisku zarządzania TickIT, które wymaga certyfikacji procesów kontroli jakości (Quality Assurance – QA) zgodnie z normą ISO 9001. Obecnie, jako współzałożyciel i dyrektor generalny firmy Original Software zajmującej się opracowywaniem narzędzi do automatyzacji testowania, George Wilson propaguje procedury elastyczne jako najlepszy sposób na osiągnięcie wysokiej jakości.

„Projekty realizowane zgodnie z metodyką Agile stanowią dla działu kontroli jakości (QA) doskonałą okazję do przejęcia przewodnictwa nad procesami [testowania]" – powiedział Wilson. Radzi on, aby testerzy objęli przywództwo zamiast trzymać się z tyłu i pozwolić programistom grać pierwsze skrzypce. „Kto ma najlepsze warunki do tego, aby wypełnić lukę między użytkownikami i programistami, zrozumieć wymagania, określić metody ich spełniania i zapewnić sukces jeszcze przed wdrożeniem?". W tym celu zespół ds. kontroli jakości musi sam działać wyjątkowo elastycznie, w związku z czym Wilson przestawia następujące fakty demaskujące niektóre z typowych mitów dotyczących elastyczności testowania:

Mit pierwszy

Trzeba tylko wykonywać testy jednostkowe – testowanie w ramach podejścia TDD wystarczy

Podejście TDD (Test Driven Development – programowanie sterowane testami) stanowi dobry początek, ale tylko dla tych, którzy nie znają innych możliwości. „Dla zdecydowanej większości komercyjnych projektów programistycznych po prostu jest to nieprawda. Nawet zdecydowani orędownicy programowania elastycznego dostrzegają potrzebę posiadania w swoim arsenale szerokiej gamy technik testowania, w tym testów typu białej i czarnej skrzynki, testów regresyjnych, obciążeniowych i odbiorczych" – powiedział Wilson.

Mit drugi

Można ponownie wykorzystywać testy jednostkowe w celu utworzenia pakietu testów regresyjnych.

Czy kiedykolwiek usłyszeli Państwo od kogoś, że konwencjonalne testy w kolejnych etapach realizacji projektów są niepotrzebne, gdyż z każdą linią kodu aplikacji jest związany odpowiedni test? „Niektórzy zwolennicy podejścia TDD sugerują, że poprzez ponowne asemblowanie testów jednostkowych można wykonać wszystkie rodzaje testów, od testów odbiorczych po regresyjne" – powiedział Wilson.

Choć może to brzmieć prawdopodobnie, Wilson twierdzi, że to nieprawda, gdyż szczegółowość i cele testów jednostkowych typu białej skrzynki, opracowanych w ramach podejścia TDD służą do czegoś innego niż testy typu czarnej skrzynki przeprowadzane w późniejszych etapach. „Podczas gdy celem ogólnym testu jednostkowego jest udowodnienie, że kod działa zgodnie z oczekiwaniami, celem testów regresyjnych jest zapewnienie, że zmiany wprowadzane w kodzie aplikacji nie spowodują żadnych nieoczekiwanych efektów. Te dwa cele nie są równoznaczne. Sprawdzenie, czy atrybut ma [na przykład] prawidłowy format daty, to nie to samo, co sprawdzenie, czy dla danego parametru wejściowego wartość pola zawiera oczekiwaną datę".

Mit trzeci

Nie potrzebujemy już testerów ani narzędzi do automatyzacji

To nieprawda. Testerzy odgrywają „inną, ale równie ważną rolę, jak programiści", jak powiedział Wilson, i dlatego powinni zajmować równorzędną pozycję w strukturze organizacyjnej projektu. „Jest [również] powszechnie wiadome, że tradycyjne narzędzia do automatyzacji testowania nie spełniają obietnic składanych przez producentów w przesadnych reklamach". Być może producenci (bez urazy, Original) powinni trochę spuścić z tonu w tych reklamach.

Mit czwarty

Testy jednostkowe eliminują potrzebę przeprowadzania testów ręcznych

Mało kto zaprzeczy, że testowanie ręczne jest powtarzalne, kosztowne, nudne i podatne na błędy. Wiadomo też, że choć podejście TDD może zmniejszyć ilość pracy ręcznej wykonywanej podczas testów funkcjonalnych, nie wyeliminuje potrzeby dalszych ręcznych lub zautomatyzowanych testów typu czarnej skrzynki. „Dzięki automatycznemu rejestrowaniu procesu realizowanego przez testera i dokumentowaniu uderzeń w klawisze i kliknięć myszą, tester będzie mieć więcej czasu na interesujące, wartościowe działania, takie jak testowanie złożonych scenariuszy, które trudno byłoby lub których nie dałoby się w ogóle wykonać automatycznie. Choć testowanie ręczne jest czasochłonnym (i dlatego kosztownym) sposobem znajdywania błędów, koszty ich nieznalezienia są jednak często znacznie wyższe" – powiedział Wilson.

Mit piąty

Testy odbiorcze nie są już potrzebne

Na podstawie swoich doświadczeń w zakresie programowania elastycznego Wilson twierdzi, że testy odbiorcze uważane są często raczej za współpracę z klientem w celu odrzucenia niewłaściwych wymagań niż za korygowanie funkcji, które nie odpowiadają potrzebom użytkowników. Niewykluczone, że prawda leży pośrodku. „Definiując na początku swoje wymagania, użytkownicy robią to na podstawie swoich początkowych wymagań. Potem, gdy widzą system »w akcji«, zawsze zgłaszają inne lub dodatkowe wymagania" – powiedział Wilson.

Proces programowania elastycznego może sprawić, że będzie się to zdarzać rzadziej. Wilson uważa jednak, że nie ma sensu oczekiwać, że podejście to całkowicie rozwiąże opisany problem. Dlatego też nie ma co liczyć na to, że uda się uniknąć testów odbiorczych. „Potwierdza się to szczególnie w przypadkach, gdy użytkownicy instytucjonalni mają dokonać odbioru interfejsu użytkownika, ponieważ mogli sobie coś wyobrażać inaczej niż programista".

Mit szósty

Automatyzacja jest niemożliwa

„Automatyzacja na wczesnych etapach projektu realizowanego zgodnie z metodyką Agile jest zazwyczaj bardzo trudna, ale w miarę rozbudowy i ewolucji systemu niektóre aspekty stabilizują się" – powiedział Wilson. W ten sposób wdrożenie automatyzacji, która pozwala zapanować nad zmianami, staje się prostsze. „Na początku wszystkie testy trzeba przeprowadzać ręcznie, ale ta praca włożona w testowanie i projektowanie może przynieść korzyści później, jeśli się ją zarejestruje i wykorzysta ponownie. Określenie właściwego czasu wprowadzenia automatyzacji jest trudne, tak więc kluczowe znaczenie ma stosowanie technologii, która będzie z wyprzedzeniem wspierać wczesne testy realizowane ręcznie, ale jednocześnie stworzy podstawy do ich przekształcenia w testy zautomatyzowane".

Mit siódmy

Programiści mają odpowiednie umiejętności w zakresie testowania

„Gdyby testowanie było łatwe, każdy mógłby to robić i dostarczalibyśmy perfekcyjny kod za każdym razem" – powiedział Wilson. Podobnie jak w przypadku pisarzy, którzy odnoszą korzyści z weryfikacji ich tekstów przez inne osoby patrzące na te teksty świeżym okiem, zespół testerów niezależnych od programistów ma lepsze spojrzenie na całość i może sprawdzić nie tylko poprawność funkcjonalną, ale również jakość materiałów przedstawianych do odbioru. „Podczas gdy programiści mają skłonność do sprawdzania tylko zgodności funkcji systemu z wymaganiami, dobry tester będzie wystarczająco bezstronny, aby zapytać »A co się stanie, jeśli...?«".

Mit ósmy

Testy jednostkowe odpowiadają 100% specyfikacji projektowych

Niezależnie od metody programowania przed opracowaniem kodu trzeba znać wymagania. „Dobrze opracowane testy przygotowane zgodnie z podejściem TDD mogą stanowić znaczny procent specyfikacji projektu, wciąż jednak problemem są luki między testami jednostkowymi" – powiedział Wilson. Uważa on, że istnieją inne równie dobre podejścia i że nie potwierdziło się w praktyce stwierdzenie, iż programista musi zastosować podejście TDD, aby udowodnić, że wszystkie wymagania zostały uwzględnione w sposób właściwy.

„Definiowanie testów sprawdzających dokładność i zwięzłość realizacji wymagań nie jest niczym nowym. Model V, na przykład, to dobre podejście ułatwiające zrozumienie wymagań w zakresie testowania, a w konsekwencji również wymagań funkcjonalnych. Podobnie jak podejście TDD, model V i większość innych modeli zawodzi, gdy podejście praktyka jest rygorystyczne, a procesy programowania płynne. Inżynieria programowania różni się od projektowania maszyn i próby wymuszania dostosowywania się do takich zasad są skazane na porażkę" – powiedział Wilson.

Niezależnie od podejścia wybranego przez zespół, należy się odnieść do każdego wymagania użytkownika, pytając, jak należałoby je przetestować. „Ważnym czynnikiem jest to, że testy należy opracować przed ich zastosowaniem względem kodu. W przeciwnym razie programistów czeka wprowadzanie wielu zmian w kodzie, czyli tzw. refaktoryzacja (refactoring)". Jeśli uszczegóławia się wymagania na drodze współpracy, „programista otrzymuje [lepszą] specyfikację, która została otwarcie oceniona z punktu widzenia różnych osób, a to oznacza, że mniej prawdopodobna będzie konieczność wprowadzania w niej zmian".

Mit dziewiąty

Podejście TDD da się zastosować w każdym projekcie

W miarę rozbudowy projektu wydłuża się również czas niezbędny do przeprowadzenia testów. Wilson twierdzi, że problem ten można rozwiązać, dzieląc na mniejsze części projekt, testy albo jedno i drugie. „Każde z tych rozwiązań powoduje powstawanie testów, które przeprowadza się z różną częstotliwością, zależnie od ich powiązań z obecnie tworzonym kodem". Takie podejście wprowadza według Wilsona potrzebę planowania testów i zarządzania ich przeprowadzaniem.

Mit dziesiąty

Programiści i testerzy są jak oliwa -i woda, nie da się ich połączyć

Pomiędzy programistami i testerami od zawsze istniały podziały typu „my i oni", ale nie musi to być przyczyną kontrowersji. „Jest to zazwyczaj zdrowa relacja symbiotyczna, która – jeśli funkcjonuje prawidłowo – zapewnia współpracę korzystną dla obu stron, która daje w efekcie wyższą jakość produktu dostarczanego klientowi." – powiedział Wilson.

Aby pomóc zapanować nad napięciami między tymi dwiema grupami, Wilson proponuje skoncentrować dyskusje na następujących zagadnieniach:

- na osiąganiu celów biznesowych, a NIE na tym, kto jest właścicielem których części procesu;
- na angażowaniu testerów już na etapie gromadzenia wymagań i procesu TDD;
- na jak najszerszym ponownym wykorzystywaniu zasobów testowych utworzonych w fazie programowania;
- na roli tradycyjnego testera w podejściu TDD i na tym, jak mogą oni zdobywać nowe umiejętności oraz w jaki sposób umożliwić im adaptację;
- na tym, jak programiści i testerzy mogą osiągać obustronne korzyści ze stosowania nowych osiągnięć w zakresie narzędzi do tworzenia i testowania oprogramowania.

Źródło:

http://www.sdtimes.com/AGILE_TESTING_FACT_AND_FICTION/By_Edward_J__Correia/32884

http://www.sdtimes.com/FIVE_MORE_MYTHS_OF_AGILE_TESTING/By_Edward_J__Correia/32921