



Magazine

Jakie wyzwania stawiają przed testowaniem metodologie Agile

Autor: Rex Black

O autorze:

Rex Black jest Prezesem RBCS (www.rbc-us.com), lidera w testowaniu oprogramowania, sprzętu i systemów. Przez ponad 10 lat RBCS dostarcza usługi konsultingowe, outsourcingowe i szkoleniowe w testowaniu oprogramowania i sprzętu. RBCS zatrudnia najbardziej doświadczonych i rozpoznawalnych konsultantów i przeprowadza z ich pomocą testowanie produktów, wydań oraz doskonalenie grup testowych i zatrudnianie personelu testowego dla setek klientów na całym świecie.

Jako lider RBCS, Rex jest obecnie najbardziej płodnym autorem praktykującym w obszarze testowania oprogramowania. Jego pierwsza popularna książka, „Zarządzanie Procesem Testowym”, została sprzedana w 35 000 egzemplarzy na całym świecie, włączając w to wydania japońskie, chińskie i hinduskie. Kolejne pięć książek o testowaniu: „Zaawansowanie Testowanie Oprogramowania: Część I”, „Zaawansowanie Testowanie Oprogramowania: Część II”, „Krytyczne Procesy Testowe”, „Podstawy Testowania Oprogramowania” i „Pragmatyczne Testowanie Oprogramowania”, również sprzedały się w dziesiątkach tysięcy egzemplarzy, łącznie z edycjami hebrajskimi, hinduskimi, japońskimi, chińskimi i rosyjskimi. Napisał ponad 25 artykułów, zrealizował setki warsztatów i seminariów, wygłosił około 30 prelekcji na konferencjach na całym świecie. Rex był Prezesem International Software Testing Qualifications Board (ISTQB) oraz dyrektorem American Software Testing Qualifications Board (ASTQB). Obecnie zasiada w radzie kierowniczej ASTQB.

Intermediate

Level

3

Magazine Number

Jakość w projekcie

Section in the magazine

Wprowadzenie

Pewna grupa naszych klientów przyswoiła Scrum i inne metodologie Agile. Każdy model cyklu życia wytwarzania oprogramowania, od sekwencyjnego poprzez spiralny, przyrostowy, do Agile, ma wpływ na testowanie. Niektóre z tych konsekwencji ułatwiają proces testowania. Tymi implikacjami nie musimy się tu przejmować.

Niektóre z tych implikacji stanowią wyzwanie w procesie testowania. W tym studium przypadku omówione zostaną owe wyzwania, tak by nasi klienci mogli poznać problemy stworzone przez metodologię Scrum i odróżnić je od innych rodzajów problemów z testowaniem, z którymi się spotykają.

W moich książkach i podczas konsultacji zwykle rekomenduję mieszaną strategię testowania, składającą się z trzech typów strategii:

- Testowanie oparte na analizie ryzyka;
- Zautomatyzowane testowanie regresyjne;
- Testowanie reaktywne (zwane również *testowaniem dynamicznym*).

Mieszana strategia, którą polecam, dobrze pasuje do Scrum i innych metodologii Agile. W niektórych przypadkach strategia ta złagodzi ryzyka testowe i zmniejszy związane z tymi metodologiami wyzwania stawiane testowaniu. Jednak – nie rozwiąże wszystkich ryzyk oraz wyzwań. W tym artykule zbadamy niektóre z wyzwań, które zauważyłem u klientów RBCS używających Scrum i metodologii Agile.

Obsługa liczby i szybkości zmian

Jedną z zasad wytwarzania Agile jest to, że zespoły projektowe powinny “przyjmować zmieniające się wymagania, nawet w późnej fazie developmentu” (agilemanifesto.org). Wiele strategii testowania, szczególnie testowanie oparte na analizie wymagań, staje się w takich sytuacjach dość nieefektywne.

Jednakże testowanie oparte na ryzyku wspiera zmianę, ponieważ zawsze możemy dodać ryzyka, usunąć je, zmienić oraz dopasowywać poziom ryzyka. Jeśli wykonywanie testów już się rozpoczęło, możemy dostosować nasz plan na pozostały okres w oparciu o ten nowy widok ryzyka jakościowego. Ponieważ testowanie oparte na ryzyku pozwala w inteligentny sposób decydować, co testować, jak bardzo i w jakiej kolejności, możemy zawsze dokonać przeglądu tych decyzji opierając się na nowych informacjach lub wskazówkach od zespołu projektowego.

Mądrze zautomatyzowane testy również wspierają bieżące zmiany. Z należytą uwagą można utrzymywać automatyzację na poziomie graficznego interfejsu użytkownika. Automatyzacja na poziomie stabilnych interfejsów opartych o komendy nie powinna dostarczyć poważniejszych problemów z utrzymaniem.

Testowanie reaktywne, które rekomenduję, nie wymaga zbyt wiele dokumentacji, jest również dość sprężyste w obliczu zmiany.

Co ważne, każda zmiana może narzucać niezależne od zastosowania tych strategii wyzwania dla testowania. Wiele z tych wyzwań wynika ze zmiany w definicji produktu i jego prawidłowego zachowania (patrz również poniżej). Jeśli zespół testowy nie jest informowany o takich zmianach, lub kiedy współczynnik zmian jest bardzo wysoki, może to spowodować nieefektywność tworzenia, wykonywania i utrzymywania testów.

Utrzymywanie efektywności podczas bardzo krótkich iteracji

W cyklach sekwencyjnych przed rozpoczęciem wykonywania testów systemowych zespoły testowe mogą mieć długi okres, w którym równoległe z developmentem systemu tworzą i utrzymują swoje testy. Niektóre, bardziej formalne iteracyjne modele cyklu życia, takie jak *Rapid Application Development* i *Rational Unified Process*, często pozwalają na znaczne okresy czasu pomiędzy okresami wykonywania testów dla każdej iteracji. Te interwały umożliwiają zespołom testowym stworzenie i utrzymywanie systemów testowych.

Metodologie Agile, takie jak Scrum, są mniej formalne i szybsze. Zgodnie z sugestywną nazwą, *sprint*, metodologie te używają krótkich iteracji o szybkim tempie. Konsultanci RBCS widzieli u pewnej liczby swoich klientów, jak to tempo i zwięzłość dusi możliwości zespołu testowego w tworzeniu i utrzymywaniu systemów testowych, pogarszając efekty zmian opisane wcześniej. Szczególnie wrażliwe na to wyzwanie okazały się strategie testowe zawierające elementy automatyzacji.

W tej sytuacji może pomóc oparty na ryzyku element rekomendowanej strategii. Testowanie oparte na ryzyku skupia się na ważnych obszarach pokrycia testów a zmniejsza, lub nawet obcina, mniej ważne obszary, zwalniając część presji stworzonej przez krótkie iteracje. Ta możliwość skupienia okazuje się szczególnie pomocna dla zespołów testowych również w przypadku ograniczeń zasobów. Zespoły testowe w świecie Agile powinny tworzyć, utrzymywać i wykonywać testy w kolejności określonej priorytetami ryzyka. Stosowanie priorytetów ryzyka do sekwencjonowania prac wytwórczych i utrzymaniowych umożliwia zespołowi testowemu ukończenie najważniejszych testów na początku okresu wykonywania testów, w każdym sprincie.

Przyjmowanie kodu po niespójnym i często nieadekwatnym testowaniu jednostkowym

Wielu autorów, praktyków i naukowców akademickich zaangażowanych w metodologie Agile podkreśla dobre, zautomatyzowane testowanie jednostkowe. Open-source'owe *jarzma testowe* (ang. *test harness*), takie jak J-Unit i Cpp-Unit minimalizują koszt narzędziowy jego wykonywania, obchodząc jedną z kluczowych przeszkód automatyzacji. Takie zautomatyzowane testy jednostkowe umożliwiają coś, co praktycy Agile nazywają *refactoringiem*. Refactoring jest przeprojektowaniem głównych kawałków kodu, lub nawet całych obiektów. Automatem testy jednostkowe umożliwiają wykonanie szybkich testów regresji kodu poddanego *refactoringowi*. Niektórzy praktycy Agile rekomendują włączanie automatycznych testów jednostkowych do projektowania, jak w *Wytwarzaniu Kierowanym Testami* (ang. *Test Driven Development*). Podczas gdy w teorii wygląda to obiecująco, istnieją dwa problemy, które można zaobserwować w praktyce.

Po pierwsze, testowanie jednostkowe jest ograniczone w kwestii wykrywania błędów. Capers Jones odkrył, że średnia efektywność usuwania defektów dla testowania jednostkowego wynosi od 25 do

30%¹. Oceny RBCS pokazały, że dobre testowanie systemowe wykonane przez niezależny zespół testowy osiąga około 85% efektywności wykrywania defektów. Tak więc, podczas gdy testowanie jednostkowe pomaga, głównym filtrem do zapobiegania nadmiernym usterkom pozostają testy systemowe.

Po drugie, odkryliśmy, że pod pozorem zarówno prawdziwych, jak i nie bardzo prawdziwych wymówek, wielu programistów nie tworzy automatycznych testów jednostkowych, a w niektórych przypadkach nie wykonuje w ogóle żadnego testowania jednostkowego. To stwarza olbrzymi problem dla zespołu testów systemowych, które – jak wspomniano wyżej – pozostają krytycznym filtrem usuwania błędów. Krótkie okresy wykonywania testów w sprintach Agile, porównując do projektów sekwencyjnych, oznaczają, że stopień zniszczenia spowodowany przez jedno- lub dwudniową blokadę postępu testowania z powodu wysoce awaryjnego kodu jest wyższy, niż w projektach sekwencyjnych.

Dostarczenie niestabilnego, błędnego kodu zmniejsza jedną z kluczowych korzyści rekomendowanej strategii testowania opartego na ryzyku, jaką jest odkrywanie najważniejszych defektów we wczesnej fazie wykonywania testów. Nieuchronnie prowadzi to również do wysokiego stopnia mieszania kodu podczas testowania, ponieważ bardzo dużo kodu będzie musiało się zmienić, by naprawić błędy. Ilość zmian może nawet w końcu przewyższyć możliwości najlepszego systemu automatycznych testów regresji, co następnie prowadzi do niższej efektywności zespołu testowego w wykrywaniu defektów.

Zarządzania zwiększonym ryzykiem regresji

Capers Jones obliczył, że regresja wyjaśnia około 7% błędów². Jednak w iteracyjnych cyklach życia, takich jak Scrum, kod, który działał w poprzednich sprintach zostaje wymieszany przez nowe funkcjonalności w każdym następnym sprincie. To zwiększa ryzyko regresji. Zwolennicy metodologii Agile podkreślają dobrze zautomatyzowane testowanie jednostkowe jako element zarządzania ryzykiem regresji - nieodłącznym w takiej mieszance.

Jednakże, dobre testowanie jednostkowe ma ograniczoną efektywność usuwania defektów – jak wcześniej zacytowano. Tak więc, testowanie regresywne zautomatyzowane poprzez testy jednostkowe prawdopodobnie pominie większość błędów regresji. Dlatego potrzebujemy efektywnego testowania regresywnego na poziomie testów systemowych (które mają wyższy poziom efektywności wykrywania defektów). Poprzez połączenie testowania opartego na ryzyku z automatycznym testowaniem regresywnym, zespół testowy może efektywnie zarządzać zwiększonym ryzykiem regresji.

Praca z ubogimi, zmiennymi lub brakującymi wyroczniami testowymi

Metodologie Agile dewaluują dokumentację pisaną. Ze szczególną pogardą traktowana jest specyfikacja. Dla przykładu, Manifest Agile sugeruje, że ludzie powinni cenić „działające oprogramowanie ponad całościową dokumentacją”. Dla zespołu testowego tworzy to prawdziwe wyzwania. Testerzy używają specyfikacji wymagań i innych dokumentów jako *wyroczni testowych*; jako środków do określenia prawidłowego zachowania w danych warunkach testu. Widzieliśmy testerów w projektach Agile, którym dano dokumenty z niewystarczającą ilością szczegółów – lub którym, w kilku przypadkach, w ogóle nie dano takich dokumentów.

¹ Zobacz artykuł Jonesa “Measuring Defect Potentials and Defect Removal Efficiency,” dostępny w Basic Library pod adresem www.rbc-us.com.

² Zobacz dla przykładu rysunki Jonesa w *Estimating Software Costs, 2e*.

Nawet jeśli zespół projektowy dostarczy zespołowi testowemu adekwatnej dokumentacji, dwie inne zasady wytwarzania zwinnego podtrzymują przy życiu wyzwanie związane z wyroczniami testowymi. Po pierwsze, Agile wymaga od zespołów przyjmowania zmian – jak omówiono wcześniej. Po drugie, zasady Agile mówią, że „najbardziej efektywną i wydajną metodą przekazywania informacji do i wewnątrz zespołu developerskiego jest rozmowa twarzą w twarz” (patrz: agilemanifesto.org). Dla wielu z naszych klientów stosujących metodologię Agile, takie jak np. Scrum, te dwie zasady pozwalają zespołowi projektowemu w każdej chwili zmienić definicję prawidłowego zachowania – nawet po tym, jak testerzy wykonali testy potwierdzające określone zachowanie, raportując przy tym dotyczące go błędy. Co więcej, definicja prawidłowego zachowania może się zmienić podczas spotkania, czy dyskusji, które nie angażowały zespołu testowego i nie musiały stworzyć udokumentowanego zapisu danej zmiany.

Żadna znana strategia testów nie może poradzić sobie z tym wyzwaniem. Rozwiązanie go wymaga zmiany zarządzania. Mierzalny symptom tego wyzwania daje procent raportów odrzuconych błędów. Raporty odrzuconych błędów są jednymi, których zespół w końcu się pozbywa, ponieważ opisują prawidłowe zachowanie. Projekty z dobrze zdefiniowanymi, stabilnymi wyroczniami testowymi cieszą się współczynnikiem odrzuconych błędów poniżej 5 procent. Projekty z ubogimi, zmieniającymi się lub brakującymi wyroczniami testowymi często wykazują współczynnik odrzuconych błędów na poziomie 30 procent i więcej.

Oszacowaliśmy narzuconą przez takie problemy z wyrocznią testową nieefektywność zespołu testowego na około 20 do 30 procent. Co więcej, brak możliwości precyzyjnego określenia, czy test dał wynik negatywny ma wpływ zarówno na efektywność testowania, jak i efektywność wykrywania defektów. Kiedy testerzy marnują czas na izolowanie sytuacji, które zespół projektowy ostatecznie definiuje jako prawidłowe działanie, zabiera to czas, który mogliby przeznaczyć na znajdowanie i raportowanie prawdziwych błędów. Te błędy tworzą następnie problemy dla klientów, użytkowników i personelu wsparcia technicznego oraz rozpraszają developerów i zespoły testowe.

Co więcej, taka sytuacja doprowadza testerów do frustracji, która obniża ich morale oraz, konsekwentnie, ich efektywność. Testerzy chcą produkować prawdziwą i prawidłową informację. Kiedy wiele z informacji, jaką dostarczają – w formie raportów odrzuconych błędów – kończy w symbolicznym koszcie na śmieci projektu, to powoduje, że ludzie zaczynają się zastanawiać, dlaczego testerzy zwracają im głowę.

Ważne jest uświadomienie sobie, że to zmniejszenie efektywności testów, wydajności i morale jest potencjalnym efektem ubocznym metodologii Agile. Organizacje używające metodologii Agile muszą w odpowiednim miejscu przypisać odpowiedzialność za takie sytuacje. Złe problemy mogą stać się o wiele gorsze, kiedy zespół testowy jest rozliczany za wyniki, które są poza ich kontrolą.

Postępowanie ze zmiennymi podstawami testów

Strategie testowania oparte na wymaganiach nie mogą poradzić sobie z niejasnymi lub brakującymi specyfikacjami wymagań. Brakujące specyfikacje wymagań mogą oznaczać, że zespół testowy przestrzegający strategii testowania opartej na wymaganiach nie tylko nie może powiedzieć, co dla poszczególnego testu oznacza, iż da on wynik negatywny czy pozytywny, ale nie będzie miał *podstaw testów*. Podstawa testów jest tym, na czym opierają się testy. Strategie testowania opartej na wymaganiach wymagają od zespołów testowych tworzenia przypadków testowych zaczynając od analizy warunków testów w wymaganiach, a następnie wykorzystania tych warunków do zaprojektowania i implementacji przypadków testowych. Brakujące lub ubogie specyfikacje wymagań nie będą odpowiednie dla takiej analizy.

Podstawy testów dostarczają też środków do mierzenia wyników. W strategii testów opartej na wymaganiach, testerzy nie mogą dokładnie raportować wyników testów, jeśli wymagania są ubogie lub ich brakuje. Testerzy nie mogą określić procentu podstaw testów pokrytych wykonanymi z wynikiem pozytywnym testami, ponieważ wymagania nie dostarczają wystarczającej ilości szczegółów do sensownej analizy pokrycia.

Z rekomendowaną tu strategią testów opartych na ryzyku, zespoły testowe mogą obejść oba problemy. Dla podstaw testów testerzy używają elementów ryzyka jakościowego. W oparciu o te elementy ryzyka jakościowego projektują i implementują przypadki testowe. Poziom ryzyka powiązanego z każdym elementem ryzyka determinuje liczbę przypadków testowych i priorytety przypadków pochodzących od danego elementu. Zespół testowy może raportować wyniki testów jako „złagodzone ryzyka jakościowe” kontra „nie złagodzone”.

Od szczegółowej dokumentacji do wielu spotkań

Jak mówi cytowany wcześniej fragment Manifestu Agile, ludzie powinni skupić się raczej na tworzeniu działającego oprogramowania, niż wszechstronnej dokumentacji. Jednakże informacja, która była – pod *odwiecznym reżimem* – zdobywana i wymieniana w tych dokumentach musi gdzieś płynąć – więc zwolennicy Agile promują „konwersacje twarzą w twarz”. Jest to oczywiście inna nazwa spotkania. Z perspektywy marketingowej, było sprytnym ze strony zwolenników Agile, nie używać słowa „spotkanie” w Manifestie Agile, ale rzeczywistość pozostaje, jaka jest.

W poprzedniej sekcji o problemach z wyrocznią testową w metodologiach Agile wspomniałem o problemie ze spotkaniem czy dyskusją na temat zmian w definicji prawidłowego zachowania, które nie angażują zespołu testowego i które nie produkują udokumentowanego zapisu zmian. W pewnych organizacjach inną stroną tego problemu jest to, że wszyscy są zapraszani na każde spotkanie, spotkania rosną w liczbę i czas trwania, menedżerowie i kierownicy nie są dostępni do zarządzania i kierowania swoimi zespołami, ponieważ przez większą część dnia przebywają na spotkaniach – przez co efektywność oraz wydajność spada.

Jeden z menedżerów tak żartobliwie opisał tę sytuację: „Scrum to ciężki proces. Jestem zaskoczony nazwą *Agile* - to powinno mieć nazwę „*leń spędzający większość czasu na kanapie*” (ang. *couch potato*). Jest za dużo spotkań. Jest za dużo gadania i śmiechu. To, że istnieją te wszystkie książki wyjaśniające, jakie to proste, jest dla mnie naprawdę ironiczne”.

Aby być uczciwym – za dużo spotkań to problem, który może dotknąć jakikolwiek projekt, realizujący jakikolwiek model cyklu życia. Pracowałem jako Menedżer Testów w klasycznych projektach wodospadowych, gdzie spędzałem 4 godziny dziennie na spotkaniach. Miałem klienta, który opowiedział komiczną anegdotę: starszy menedżer, w odpowiedzi na skargę od *line managera* o tym, jaki negatywny wpływ na jego możliwości kierowania zespołem miało uczęszczanie na spotkania, wrzasnął: „Będziemy kontynuować te spotkania dopóki nie dowiem się, dlaczego nic nie jest tu zrobione!”.

To pokazuje, że każda organizacja, każdy projekt i każdy cykl życia musi osiągnąć właściwą równowagę. W niektórych przypadkach, organizacje i projekty przestrzegające metodologii Agile reagują zbyt silnie na komentarze dotyczące dokumentacji i „dyskusji” twarzą w twarz z Manifestu Agile. Co więcej, przyjmowanie zmian nie powinno powodować odrzucania lub ponownego rozważania poprzednich decyzji do tego stopnia, że paraliżuje to zespół. Zespoły używające metodologii Agile muszą osiągnąć właściwą równowagę pomiędzy dokumentacją a spotkaniami. Muszą też mieć krótkie, efektywne spotkania, które prowadzą projekt naprzód i są zgodne z obranym kursem; nie spotkania, które wykańczają zespół i kręcą się w kółko.

Zaufanie do umownych czasów trwania iteracji

Niektórzy z naszych klientów stosujących metodologie Agile, takie jak Scrum, mają skłonność do rytualizacji niektórych zasad procesu. Szczególnym przedmiotem takich rytuałów są terminy iteracji. Po pierwsze, byłem zaskoczony tym, że ci sami klienci odrzucają pewne inne zasady, takie jak wymaganie testowania jednostkowego, często z mniej ważnych powodów, niż powody, dla których ściśle trzymają się terminów. Terminy są jednakże namacalne, podczas gdy korzyści z testów jednostkowych nie są tak dobrze rozumiane i jasne, mogą więc teraz zrozumieć przyczyny leżące poza wybiórczym wyróżnianiem określonych zasad procesu Agile ponad innymi.

Dla przykładu, załóżmy, że zespół projektowy stosuje 4-tygodniowe sprinty. Systemowym problemem w naszej branży związanym z szacowaniem oprogramowania jest tendencja do przeszacowania liczby funkcjonalności (*user story*) dla danego sprintu. Tak więc, ostatniego piątku danej iteracji, z developmentem kończącym się późno, umowny termin pozostaje nienaruszony – kosztem pracy zespołu testowego w weekend.

Pełne rozwiązanie tego wyzwania wymaga dojrzałości zespołu i zarządzania. Kiedy ludzie notorycznie i systematycznie przeszacowują, kierownictwo powinno wymagać użycia historycznych metryk postępu do szacowania. Wielu praktyków Agile używa takich metryk, jak wykresy wypalania (ang. *burndown charts*) lub prędkość *story-point* (ang. *story-point velocity*). Proces Scrum obejmuje gromadzenie i wykorzystywanie takich metryk.

Jeśli nie można naprawić problemów z szacowaniem oprogramowania, testowanie oparte na ryzyku pomaga zespołowi testowemu poradzić sobie z systematycznym i stałym przeszacowywaniem. Zaczynając od tego, że kiedy pod koniec sprintu zespół testowy po raz kolejny znajduje się w sytuacji kryzysowej, powinien zaakceptować fakt, że priorytety zespołu projektowego wynikają z harmonogramu, nie kierują się jakością. Zespół testowy powinien zweryfikować podejście do analizy ryzyka, aby w kolejnych projektach podczas analizy ryzyka w całym zakresie wdrożyć redukcję zakresu testowania przypisanego do każdego elementu ryzyka jakościowego. Tym sposobem, przynajmniej zespół testowy nie będzie przeszacowywał.

W niektórych przypadkach pomimo zmniejszania zakresu testowania, na końcu iteracji zespół testowy nadal nie może wykonać wszystkich testów w dostępnym czasie.

Jeśli tak, to zamiast marnować weekend na wykonanie wszystkich testów, zespół testowy może - wykorzystując numer priorytetu ryzyka - wybrać najważniejsze testy. Mniej ważne testy mogą być przełożone na następną iterację. (Zauważycie, że Scrum i inne metodologie Agile w ten sam sposób umożliwiają przesuwanie *user story* z jednego sprintu do następnego). Oczywiście, jeśli zespół testowy musi stale w taki sposób selekcjonować swoje testy, powinien ponownie dostosować mapowanie zakresu testów dla przyszłych iteracji.

Postępowanie z martwymi strefami w silosie iteracji

Nie wszyscy eksperci metodologii Agile zgadzają się co do potrzeby niezależnych zespołów testowych. Niektórzy zdają się myśleć o testowaniu jako o podzbiórce zadań wykonywanych przez programistów w zespole Agile, w szczególności tworzeniu różnych form automatycznych testów jednostkowych i/lub

testów akceptacyjnych. Może to wyłączać potrzebę istnienia niezależnego zespołu testowego, lub może zmieniać rolę takiego zespołu³.

Większość z naszych klientów adaptujących metodologie Agile zachowała w pewnym zakresie niezależny zespół testowy, lub co najmniej niezależnego testera. Wśród tych, którzy zachowali niezależny zespół, większość wybrała podzielenie zespołu pomiędzy iteracjami w pewien sposób, często czyniąc każdego testera raportującym codzienne zadania raczej do ScrumMastera lub innego lidera iteracji, niż do menedżera testów. W przypadku posiadania niezależnych testerów, ale nie niezależnego zespołu testowego, rola menedżera testów nie ma zastosowania.

Daje to pewne korzyści, szczególnie dla osoby kierującej iteracją:

- Każdy tester skupia się całkowicie na zadaniach związanych z iteracją, z minimalnym rozproszeniem z zewnątrz.
- Każdy tester przeznacza czas całkowicie dla korzyści iteracji i jej pilnych celów.
- Lider iteracji może przekierować któregośkolwiek testera, by skupił się na tym, co najważniejsze dla zespołu iteracji, często bez konieczności konsultacji z menedżerem testów.
- Lider iteracji może – i na końcu iteracji często to robi – wezwać testera i dać mu dodatkowe zadania, by osiągnąć cele sprintu.
- Ilość prac testowych przeznaczonych na iterację nie zmienia się po tym, jak określono liczbę testerów dla tej iteracji - menedżer testów nie może zaskoczyć lidera iteracji nagłą redukcją prac testowych, by przysłużyć się innym priorytetom zespołu testowego.

Jak można zauważyć, niektóre z tych korzyści bywają elementami gry o sumie stałej (ang. *zero-sum game*), które niosą w sobie źródła – o ile nie są nimi rzeczywiście – potencjalnych problemów.

Wyzwania tego podejścia – z których niektóre są oczywistymi następstwami korzyści – obejmują następujące:

- Tester myśli o sobie jako – i postępuje w ten sposób – mniej o testerze a bardziej o wyspecjalizowanym developerze. Może to powodować utratę zainteresowania do rozwijania umiejętności specyficznych dla testowania na rzecz doskonalenia umiejętności technicznych.
- Tester ma mniejszy kontakt z ludźmi spoza zespołu iteracji, co redukuje szerszą perspektywę poziomu systemowego, dostarczaną przez niezależny zespół testowy.
- W przypadku braku wskazówek koordynujących od menedżera testów, tester zaczyna popełniać błędy związane z lukami i kolizjami. Tester odnosi porażki w wykonywaniu niektórych zadań testowych, które mają sens – szczególnie dotyczy to długoterminowych inwestycji, które niekoniecznie przynoszą korzyść w aktualnej iteracji. Tester nadmiarowo wykonuje zadania testowe zrealizowane przez innych testerów w innych iteracjach, ponieważ nie jest świadomy pracy innych.

³

Dla przykładu zobacz streszczenie przemowy Kenta Becka, wygłoszonej na konferencji Tydzień Jakości 2001, www.soft.com/QualWeek/QW2001/papers/2Q.html. Beck stwierdził, że testowanie jednostkowe wykonywane przez programistów ewentualnie może uczynić zbędną rolę niezależnego zespołu testowego (nazwanego „QA” w tym streszczeniu) w wykrywaniu defektów, przekształcając testowanie w rolę podobną do roli analityka biznesowego. Moje wcześniejsze komentarze odnośnie testowania jednostkowego i rysunki ograniczeń efektywności testowania jednostkowego, które przytaczałem z prac Capera Jonesa, powodują, że zaczynam być sceptyczny odnośnie tego, że zobaczymy istotnie wolny od błędów kod dostarczony zespołom testowym, niezależnie od tego, jaki jest model cyklu życia – ale cieszyłbym się pracując w projektach, w których ktoś dałby radę udowodnić mi, że się mylę.

- Menedżer testów nie mający możliwości zarządzania obciążeniem zadaniami swoich zasobów, odkrywa, że morale zespołu cierpi i zwiększa się fluktuacja, ponieważ niezrównoważone obciążenie pracą na końcu każdego sprintu, zabiera testerom narzędzia pracy.
- Możliwości zespołu testowego w rozwijaniu spójnego, potężnego, utrzymywalnego systemu testowego – skryptów testowych, narzędzi testowych, danych i innych rzeczy niezbędnych w długoterminowej perspektywie do efektywnego i wydajnego testowania – zmniejszają się z powodu pilnego skupiania się na natychmiastowych potrzebach iteracji.
- Testerzy wykazują tendencję do „bycia swoimi” i tracą pewien obiektywizm, który normalnie daje niezależny zespół testowy, powodując zmniejszenie efektywności wykrywania błędów.

Żadne z tych wyzwań (lub korzyści) nie wynika z samych metodologii Agile; po prostu zdarza się, że metodologie Agile jako praktykowane na co dzień mają skłonność do ich uwypuklenia. Przez lata obserwowałem podobne problemy w projektach przestrzegających modeli sekwencyjnych, lub nie przestrzegających w ogóle żadnych modeli, gdzie do organizacji zespołu testowego stosowano coś, co nazywam podejściem *zasobów projektowych* (ang. *project resource*).

Wyzwania te są do pokonania, jeśli istnieje niezależny zespół testowy. Zespół testowy kierowany przez dobrego menedżera testów może wprowadzić rozwiązania, które scalą razem zespół i uczynią jego działania spójnymi i zgodnymi. Te rozwiązania następnie równoważą specyficzne dla sprintu „siły odśrodkowe”, które mają tendencję do zmuszania członków zespołu testowego do optymalizacji pracy dla danej iteracji oraz izolowania członków zespołu testowego od szerszych rozważań na temat testowania⁴.

Jednym z podejść do zarządzania tymi wyzwaniami jest posiadanie oddzielnego okresu testowania, który jest poprzedzony iteracją developerską. (By utrzymać czystą terminologię, niektórzy lubią nazywać to podejście alternacją *sprintów developerskich śledzonych przez sprinty testowe*). Niektórzy z naszych klientów potępiali to podejście. Wydawało im się, że to działa wtedy, kiedy zespół testowy jest nadal zaangażowany w iterację developerską. W innych słowach, jeśli zespół testowy odłącza się od zespołu developerskiego podczas iteracji developerskiej, zamieniacie po prostu jedną formę grupowania na inną.

Zarządzanie oczekiwaniami

Aby zamknąć ten artykuł, przyjrzyjmy się psychologicznemu wyzwaniu dla zespołów testowych w projektach Agile. Gartner, znani analitycy rynku, mówią, że przyswajanie nowych technologii i koncepcji przez przemysł IT doświadcza *Cyklu Ekscytacji* (www.gartner.com/pages/story.php.id.8795.s.8.jsp). Cykl Ekscytacji składa się z 5 odrębnych faz, jak to opisano na stronie www:

1. **Wyzwalacz technologiczny.** Pierwsza faza Cyklu Ekscytacji to „wyzwalacz technologiczny” lub przełom, start produktu czy inne wydarzenie generujące znaczne zainteresowanie – także mediów.
2. **Szczyt rozdmuchanych oczekiwań.** W następnej fazie szal ogółu zwykle generuje nadmierny entuzjazm i nierealistyczne oczekiwania. Mogą być pewne udane zastosowania technologii, ale zwykle więcej jest porażek.

⁴ Patrz Rozdział 8 mojej książki *Managing the Testing Process*. Pierwsza edycja (1999), druga (2003) i trzecia (2009) wszystkie kładą nacisk na to samo. *Plus ça change, plus c'est la même chose...* [przypis redakcji: Im bardziej rzeczy się zmieniają, tym bardziej pozostają te same].

3. **Koryto rozczarowania.** Technologie wchodzą w “koryto rozczarowania”, ponieważ nie udaje im się spełnić oczekiwań, szybko stają się niemodne. Konsekwentnie, media zwykle porzucają temat i daną technologię.
4. **Pochylenie oświecenia.** Mimo, że media mogły przestać zajmować się technologią, niektóre biznesy kontynuują poprzez „pochylenie oświecenia” i eksperymentowanie celem zrozumienia korzyści i praktycznego zastosowania technologii.
5. **Płaskowyż produktywności.** Technologia osiąga “płaskowyż produktywności”, jako że jej korzyści zostają szeroko zademonstrowane i zaakceptowane. Technologia staje się coraz stabilniejsza i ewoluuje w generacje drugą i trzecią. Ostateczna wysokość płaskowyżu różni się w zależności od tego, czy technologia może być zastosowana szeroko, czy daje korzyści tylko rynkom niszowym.

Dotarliśmy do 2010 roku i widzimy metodologie Agile w fazie szczytu rozdmuchanych oczekiwań. Klienci RBCS mają wygórowane oczekiwania w stosunku do metod Agile dotyczących jakości, produktywności i elastyczności.

Niektóre zespoły testowe w projektach Scrum donoszą kierownictwu, że jakość produktu nie jest wyższa, niż normalnie, a czasem nawet niższa. Niektóre zespoły raportują kierownictwu, że wyzwania omówione w tym artykule zmniejszyły ich efektywność. Niektóre zespoły testowe w projektach Scrum donoszą kierownictwu, że podczas, gdy development nie doświadcza negatywnych konsekwencji lub bólu zmian – co jest kluczową obietnicą metodologii Agile i jednym z kluczowych punktów przetargowych kierownictwa – testowanie wciąż napotyka problemy związane z nieograniczonymi, niezarządzanymi zmianami.

Kiedy zespoły testowe donoszą kierownictwu o tych negatywnych wynikach, kierownictwo doświadcza psychologicznego fenomenu zwanego *dysonansem poznawczym*. Dysonans poznawczy polega na odczuwaniu mentalnego napięcia pomiędzy niekompatybilnością oczekiwań względem metodologii Agile, a zaobserwowanymi wynikami.

Ostatecznie te doświadczenia dysonansu poznawczego wśród różnych osób adaptujących metodologie Agile spowodują, że te podejścia wyjdą z Cyklu Ekscytacji, poza szczyt rozdmuchanych oczekiwań. Jednak w krótkim okresie kierownictwo może popaść w inny psychologiczny fenomen, zwany *projekcją*. Zawiera się w tym także projektowanie uwzględniające innych - jakie mamy odczucia w stosunku do tego, z czym są oni powiązani, lecz niekoniecznie za to odpowiedzialni. Doświadczeni profesjonaliści testowi lepiej znają ten fenomen pod pojęciem *zabijanie powstańca*.

Podsumowanie

Strategia testowania, którą zwykle rekomenduję będzie wspierać stwierdzone cele metodologii Agile. Testowanie oparte na ryzyku wspiera wzrost jakości, ponieważ koncentruje testowanie na obszarach wysokiego ryzyka, gdzie testy mogą znacząco zmniejszyć ryzyko. Testowanie oparte na ryzyku pomaga zwiększyć produktywność, ponieważ redukuje lub eliminuje testowanie tam, gdzie ryzyko jakościowe jest niskie. Testowanie oparte na ryzyku wspiera elastyczność, gdyż umożliwia regularne przeglądanie elementów ryzyka jakościowego, co dostosowuje pozostające testowanie do nowych ryzyk i ich nowych poziomów ryzyka. Automatyczne testy regresji pomagają zapanować nad związanym z metodologiami Agile ryzykiem regresji, dopuszczając wyższy współczynnik zmian. Reaktywne testowanie umożliwia testerom odkrywanie różnych aspektów systemu, które testowanie oparte na ryzyku wraz z automatycznymi testami regresji mogło pominąć.

Jednakże to mieszanie strategii opartej na ryzyku, automatyzacji i strategii reaktywnej nie może w pełni rozwiązać wyzwań opisanych w tym studium przypadku. W długim okresie ludzie dojdą do takich wniosków i zwyciężą racjonalne koszty alternatywne. Jednakże w krótkim okresie, do czasu gdy metodologie Agile pozostają na szczycie rozdmuchanych oczekiwań, zespół testowy musi być ostrożny w komunikowaniu wszelkich problemów z testowaniem, które wynikają bardziej z metodologii Agile oraz ich wpływu na testowanie, niż z samego testowania.