Certyfikowany Tester
Sylabus rozszerzenia poziomu podstawowego
Tester zwinny

Wersja 2014.01

International Software Testing Qualifications Board®

Prawa autorskie

Grupa robocza Foundation Level Extension Agile Tester: Rex Black (Przewodniczący), Bertrand Cornanguer (Wiceprzewodniczący),Gerry Coleman (Lider celów uczenia), Debra Fredenberg (Lider pytań egzaminacyjnych), Alon Linetzki (Lider d/s marketingu oraz korzyści biznesowych), Tauhida Parveen (Edytor), and Leo van der Aalst (Lider dewelopmentu).

Autorzy: Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh, and Stephan Weber

Przegląd wewnętrzny: Mette Bruhn Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoeckle- Olsen, Robert Treffny, Chris Van Bael, and Erik van Veenendaal;2013-2014.

Tłumaczenie na język polski: Lucjan Stapp. Przegląd tłumaczenia: Joanna Kazun, Adam Ścierski, Adam Waleska, Bartosz Bielewicz. Scalenie dokumentu i edycja: Jan Sabak. Końcowy przegląd: Lucjan Stapp, Łukasz Pardoła, Joanna Kazuń. Końcowa edycja: Lucjan Stapp

Historia zmian

Wersja	Data	Uwagi
Sylabus 2014PL.1	24.02.2021	Przegląd edytorski
Sylabus 2014PL.1	2014-11-30	Usunięcie literówek i niezgodności wskazanych przez użytkowników
Sylabus 2014PL	2014-10-24	Zatwierdzenie przez Zarząd SJSI
Sylabus v102PL	2014-10-13	Wprowadzone zmiany po przeglądzie wersji v101PL
Sylabus v101PL	2014-09-15	Wprowadzone zmiany po przeglądzie wersji v10PL
Sylabus v10PL	2014-08-15	Wprowadzone zmiany po przeglądzie wersji v09PL
Sylabus v09PL	2014-06-01	Tłumaczenie na język polski
Sylabus 2014	2014-05-31	Zatwierdzenie przez GA
Sylabus v09	2014-01-30	Wersja beta
Sylabus v071	2013-12-20	Aktualizacja przez grupę roboczą wersji v07
Sylabus v07	2013-12-16	Wprowadzone zmiany z przeglądu alfa wersji v03
Sylabus v03	2013-10-20	Wprowadzone zmiany z przeglądu wersji v02
Sylabus v02	2013-09-16	Wprowadzone zmiany z przeglądu wersji v01
Sylabus v01	2013-07-26	Oddzielne sekcje

Spis treści

Podziękowania

Dokument ten został napisany przez zespół należący do International Software Testing Qualifications Board Foundation Level Workin Group.

Zespół Rozszerzenia Zwinnego dziękuję zespołowi przeglądającemu oraz wszystkim Radom Krajowym za sugestie i wkład.

W momencie zakończenia tworzenia sylabusa Rozszerzenia Poziomu Podstawowego – Tester Zwinny grupa robocza składała się z następujących osób: Rex Black (Przewodniczący), Bertrand Cornanguer (Wiceprzewodniczący),Gerry Coleman (Lider celów uczenia), Debra Fredenberg (Lider pytań egzaminacyjnych), Alon Linetzki (Lider d/s marketingu oraz korzyści biznesowych), Tauhida Parveen (Edytor), and Leo van der Aalst (Lider dewelopmentu).

Autorzy: Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh, and Stephan Weber

Przegląd wewnętrzny: Mette Bruhn Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoeckle- Olsen, Robert Treffny, Chris Van Bael, and Erik van Veenendaal;2013-2014.

Zespół dziękuje również następującym osobom z Rad Krajowych oraz zwinnym ekspertom, którzy brali udział w przeglądaniu, komentowaniu oraz głosowaniu nad Rozszerzeniem Poziomu Podstawowego – Tester Zwinny (w porządku alfabetycznym): Dani Almog, Stephen Bird, Richard Berns, Monika Bögge, Josephine Crawford, Tibor Csöndes, Jurian Van de Laar, Huba Demeter, Marnix Van Den Ent, Arnaud Foucal, Cyril Fumery, Kobi Halperin, Inga Hansen, Hanne Hinz, Jidong Hu, Phill Isles, Shirley Itah, Martin Klonk, Kjell Lauren, Igal Levi, Rik Marselis, Johan Meivert, Armin Metzger, Peter Morgan, Ninna Morin, Ingvar Nordstrom, Chris O'Dea, Klaus Olsen, Ismo Paukamainen, Nathalie Phung, Helmut Pichler, Salvatore Reale, Hans Rombouts, Petri Säilynoja, Soile Sainio, Lars-Erik Sandberg, Dakar Shalom, Jian Shen, Shirley Silverblat, Marco Sogliani, Lucjan Stapp, Yaron Tsubery, Stephanie Ulrich, Tommi Välimäki, António Vieira Melo, Wenye Xu, Ester Zabar, Wenqiang Zheng, Peter Zimmerer, Stevan Zivanovic, Terry Zuo.

Dokument ten został formalnie zatwierdzony do wydania przez Zgromadzenie Ogólne ISTQB® dnia 31 maja 2014r.

0 Wstęp

0.1 Cel

Sylabus ten stanowi podstawę dla międzynarodowego certyfikatu w testowaniu oprogramowania na poziomie podstawowym dla testerów zwinnych. ISTQB® daje ten dokument:

- Radom Krajowym, w celu przetłumaczenia na lokalny język i akredytowania dostawców szkoleń. Rady Krajowe mogą zaadaptować ten sylabus do swoich potrzeb językowych oraz zmodyfikować referencje tak żeby zawierały lokalne publikacje.
- Ciałom Egzaminującym w celu przygotowania pytań egzaminacyjnych w lokalnych językach zaadaptowanych do celów nauczania z każdego sylabusa.
- Dostawcom szkoleń w celu przygotowania materiałów kursowych i ustalenia odpowiednich metod nauczania.
- Kandydatom do certyfikatu w celu przygotowania do egzaminu (w ramach kursów lub niezależnie od nich).
- Międzynarodowej społeczności twórców oprogramowania i systemów, w celu rozwoju zawodu tester oprogramowania i systemów oraz jako bazę dla książek i artykułów.

ISTQB® może pozwolić innym ciałom na wykorzystanie sylabusa do innych celów pod warunkiem, że wystąpią o pozwolenie na piśmie i je uzyskają.

0.2 Opis

Poziom podstawowy składa się z trzech oddzielnych sylabusów:

- Certyfikowany Tester.
- Tester Zwinny.
- Testowanie oparte na modelach (w przygotowaniu).

Dokument opisu Poziomu Podstawowego Testera Zwinnego [ISTQB_FA_OVIEW] zawiera następujące informacje:

- korzyści biznesowe dla każdego sylabusa;
- podsumowanie dla każdego sylabusa;
- powiązania pomiędzy sylabusami;
- opisy poziomów poznawczych (poziomów K);
- dodatki.

0.3 Cele nauczania podlegające egzaminowi

Cele Nauczania wynikają z Korzyści Biznesowych i są wykorzystywanie do stworzenia egzaminu na Certyfikowanego Testera Poziomu Podstawowego – Tester Zwinny. Wszystkie części sylabusa są sprawdzane na egzaminie na poziomie K1. To znaczy, czy kandydat rozpozna, pamięta pojęcie lub koncepcję. Cele nauczania na poziomach K2 i K3 zostały wypisane na początku każdego rozdziału.

1 Zwinne wytwarzanie oprogramowania - 150 minut

Słowa kluczowe

automatyzacja testów, cykl życia oprogramowania historyjka użytkownika, iteracyjny model wytwarzania oprogramowania, Manifest Agile (Manifest Zwinnego Wytwarzania Oprogramowania)
podstawa testów, przyrostowy model wytwarzania oprogramowania, wyrocznia testowa, wytwarzanie sterowane testami, zwinne wytwarzanie oprogramowania

Cele nauczania dla zwinnego wytwarzania oprogramowania

1.1 Podstawy zwinnego wytwarzania oprogramowania

FA-1.1.1 (K1) Kandydat pamięta podstawowe zasady zwinnego wytwarzania oprogramowania w oparciu o Manifest Agile.
FA-1.1.2 (K2) Kandydat rozumie korzyści wynikające z podejścia “cały zespół”.
FA-1.1.3 (K2) Kandydat rozumie korzyści wynikające z wczesnego i częstego otrzymywania informacji zwrotnej.

1.2 Aspekty podejść zwinnych

FA-1.2.1 (K1) Kandydat pamięta podejścia zwinne do wytwarzania oprogramowania.
FA-1.2.2 (K3) Kandydat potrafi napisać historyjki użytkownika we współpracy z wytwórcami, przedstawicielami biznesu i właścicielem produktu.
FA-1.2.3 (K2) Kandydat rozumie, jak spotkania retrospektywne mogą być wykorzystywane jako mechanizm do doskonalenia procesu w projektach zwinnych.
FA-1.2.4 (K2) Kandydat rozumie wykorzystanie i cele ciągłej integracji.
FA-1.2.5 (K1) Kandydat zna różnice pomiędzy planowaniem iteracji i wydania, a także wie, w jaki sposób tester dodaje wartości każdej z tych aktywności.

1.1 Podstawy zwinnego wytwarzania oprogramowania

Tester w projekcie zwinnym pracuje w odmienny sposób niż tester w projekcie tradycyjnym. Testerzy muszą rozumieć wartości i zasady, według których prowadzone są zwinne projekty oraz to, w jaki sposób stanowią integralną część podejścia „cały zespół” razem z deweloperami i przedstawicielami biznesu. Członkowie zespołów zwinnych komunikują się ze sobą jak najszybciej i często, co pomaga we wczesnym usuwaniu defektów i rozwoju produktów o wysokiej jakości.

1.1.1 Zwinne wytwarzanie oprogramowania i Manifest Agile

W 2001 roku grupa osób reprezentujących najbardziej rozpowszechnione metodyki zwinnego wytwarzania oprogramowania uzgodniła wspólny zbiór wartości i zasad. Zbiór ten znany jest

jako Manifest Zwinnego Wytwarzania Oprogramowania lub Manifest Agile [agilemanifesto.org]. Manifest Agile zawiera cztery główne zasady:

- ludzie i współpraca ponad procesy i narzędzia;
- działające oprogramowanie ponad obszerną dokumentację;
- współpraca z klientem ponad formalne ustalenia;
- reagowanie na zmiany ponad podążanie za planem.

Manifest Agile docenia pomysły wymienione po prawej stronie, jednak uznaje, iż te po lewej mają znacznie większą wartość.

Ludzie i współpraca

Wytwarzanie zwinne jest bardzo zorientowane na ludzi. Zespół złożony z konkretnych osób budujący oprogramowanie pracuje najefektywniej dzięki ciągłej komunikacji i współpracy, a nie poprzez poleganie na narzędziach lub procesach.

Działające oprogramowanie

Z punktu widzenia klienta działające oprogramowanie jest zdecydowanie bardziej użyteczne i wartościowe niż nadmiernie szczegółowa dokumentacja, ponadto zapewnia ono szanse na szybkie dostarczenie zespołowi wytwórczemu informacji zwrotnych. Co więcej, ponieważ działające oprogramowanie, nawet z ograniczoną funkcjonalnością, jest dostępne znacznie wcześniej w cyklu życia wytwarzania, wytwarzanie zwinne może mieć wcześniejszy termin wdrożenia (ang. „time to market"). W związku z tym wytwarzanie zwinne jest szczególnie użyteczne w bardzo zmiennym otoczeniu biznesowym, gdy problemy i rozwiązania są niejasne lub gdy biznes chce wprowadzać zmiany w nowym obszarze.

Współpraca z klientem

Klienci często mają duże problemy z wyspecyfikowaniem wymagań na system. Dzięki bezpośredniej współpracy z klientem, wzrastają szanse na zrozumienie, czego klient tak naprawdę potrzebuje. Jakkolwiek kontrakt z klientem może być istotny, przy regularnej i ścisłej z nim współpracy wzrasta szansa projektu na osiągnięcie sukcesu.

Reagowanie na zmiany

Zmiany są nieuniknione w większości projektów informatycznych. Środowisko, w którym działa biznes, prawodawstwo, działalność konkurencji, postęp technologiczny oraz inne czynniki mają istotny wpływ na projekt i jego cele. Proces wytwórczy musi się do nich dostosowywać. W związku z tym, zachowanie elastyczności w sposobie pracy, tak aby radzić sobie ze zmianami, jest istotniejsze niż sztywne trzymanie się planu.

Wartości

Podstawowe wartości manifestu Agile zostały ujęte w 12 wartości :

- Najważniejsze dla nas jest zadowolenie klienta wynikające z wcześnie rozpoczętego i ciągłego dostarczania wartościowego oprogramowania.
- Bądź otwarty na zmieniające się wymagania nawet na późnym etapie projektu. Zwinne procesy wykorzystują zmiany dla uzyskania przewagi konkurencyjnej przez Klienta.
- Często dostarczaj działające oprogramowanie, najlepiej w odstępach od kilku tygodni do paru miesięcy, im częściej tym lepiej.
- Współpraca między biznesem i programistami musi odbywać się codziennie w trakcie trwania projektu.

- Twórz projekty wokół zmotywowanych osób. Daj im środowisko i wsparcie, którego potrzebują i zaufaj, że dobrze wykonają swoją pracę.
- Najwydajniejszym i najskuteczniejszym sposobem przekazywania informacji do i w ramach zespołu wytwórczego jest rozmowa twarzą w twarz.
- Podstawową i najważniejszą miarą postępu jest działające oprogramowanie.
- Zwinne procesy tworzą środowisko do równomiernego rozwijania oprogramowania. Równomierne tempo powinno być nieustannie utrzymywane poprzez sponsorów, programistów oraz użytkowników.
- Ciągłe skupienie na technicznej doskonałości i dobrym projekcie oprogramowania zwiększa zwinność.
- Prostota – sztuka maksymalizacji pracy niewykonanej – jest zasadnicza.
- Najlepsze architektury, wymagania i projekty powstają w samoorganizujących się zespołach.
- W regularnych odstępach czasu zespół zastanawia się, jak poprawić swoją efektywność, dostosowuje lub zmienia swoje zachowanie.

Różne zwinne metodyki dostarczają własnych dobrych praktyk umożliwiających wprowadzenie tych wartości w życie.

1.1.2 Podejście "cały zespół"

Podejście „cały zespół" oznacza, że w celu zapewnienia sukcesu projektu angażowany jest każdy posiadający odpowiednią wiedzę i umiejętności. Zespół obejmuje przedstawicieli klienta i innych interesariuszy biznesowych, którzy określają właściwości produktu. Typowy zespół składa się na ogół z pięciu do dziewięciu osób. Sugeruje się, by cały zespół pracował w jednym miejscu, gdyż ułatwia to komunikację i współdziałanie. Podejście „cały zespół" jest wspierane przez codzienne spotkania na stojąco zwane też spotkaniami porannymi (ang. stand-up meetings; patrz sekcja 2.2.1). W te spotkania angażują się wszyscy członkowie zespołu. W trakcie tych spotkań przedstawiany jest postęp prac, a także pokazywane są wszystkie przeszkody. Podejście „cały zespół" promuje bardziej efektywną i skuteczną dynamikę zespołu.

Wykorzystanie podejścia „cały zespół" do wytwarzania produktów stanowi jedną z głównych korzyści zwinnego wytwarzania oprogramowania. Podejście „cały zespół" ma wiele zalet, między innymi:

- Poprawia współpracę i komunikację w zespole.
- Umożliwia wykorzystanie synergii umiejętności członków zespołu z pożytkiem dla całego projektu.
- Czyni wszystkich odpowiedzialnymi za jakość.

W projekcie zwinnym za jakość odpowiada cały zespół. Kwintesencją podejścia „cały zespół" jest to, że testerzy, programiści oraz reprezentanci biznesu pracują razem na każdym kroku procesu rozwoju oprogramowania. Testerzy ściśle współpracują zarówno z deweloperami jak i z przedstawicielami biznesu, by zapewnić osiągnięcie pożądanego poziomu jakości. To współdziałanie obejmuje: pomaganie i współpracę z przedstawicielami biznesu, by wesprzeć ich w tworzeniu odpowiednich testów akceptacyjnych; współdziałanie z deweloperami, by uzgodnić strategię testowania oraz decydować o podejściu do automatyzacji testów. Testerzy mogą więc przenosić i szerzyć wiedzę o testach wśród innych członków zespołu i wpływać na wytwarzanie produktu.

Cały zespół jest zaangażowany w konsultacje oraz spotkania, na których właściwości produktu są przedstawiane, analizowane lub oceniane. Koncepcja angażowania testerów, deweloperów i przedstawicieli biznesu we wszystkie dyskusje odnośnie właściwości nazywana jest siłą trzech (ang. „the power of three") [Crispin08].

1.1.3 Wczesna i częsta informacja zwrotna

Projekty zwinne mają krótkie iteracje; tym samym zespół projektowy może otrzymywać wczesne i częste informacje zwrotne dotyczące jakości produktu przez cały cykl wytwarzania. Jedną z metod szybkiego otrzymywania informacji zwrotnej jest ciągła integracja (patrz sekcja 1.2.4).

Kiedy wykorzystywane jest sekwencyjne podejście do wytwarzania, użytkownik często nie widzi wyników projektu aż do momentu, gdy cały produkt jest prawie gotowy. Na ogół jest już zbyt późno dla zespołu wytwórczego, by sprawnie rozwiązać problemy, które mogą zgłaszać użytkownicy. Gdy mamy częstą informację zwrotną od użytkowników podczas całego wytwarzania, zespół zwinny może wprowadzać te nowe informacje do procesu wytwarzania produktu. Oznacza to koncentrowanie się na właściwościach o największej wartości biznesowej lub największym ryzyku. Właściwości te są jako pierwsze przekazywane użytkownikowi. Dzięki częstej informacji zwrotnej, zespól zwinny dowiaduje się o swoich własnych możliwościach. Na przykład, ile pracy możemy zrobić w jednym sprincie lub jednej iteracji? Co przyspieszyłoby naszą pracę? Co uniemożliwia optymalizację pracy?

Zyskami z szybkiego i częstego otrzymywania informacji zwrotnej są, m.in.:

- Uniknięcie niezrozumienia wymagań, wykrycie czego w późnych etapach cyklu wytwórczego oraz poprawienie będzie dużo bardziej kosztowne.
- Wyjaśnianie zgłoszeń klienta odnośnie właściwości. Wczesne i regularne wyjaśnianie żądań podczas wytwarzania, zwiększa prawdopodobieństwo, że kluczowe właściwości będą dostępne dla użytkownika wcześniej i że produkt będzie bardziej odzwierciedlał to, co klient potrzebuje.
- Natychmiastowe ostrzeganie zespołu wytwórczego o problemach jakościowych przy wykorzystaniu procesu ciągłej integracji. Problemy mogą być łatwiej izolowane i rozwiązywane niż gdyby zostały wykryte później w cyklu wytwórczym.
- Informowanie zespołu zwinnego o jego wydajności i zdolności do dostarczenie produktu na czas.
- Promowanie stałego tempa prac projektowych.

1.2 Aspekty podejść zwinnych

Istnieje kilka technik wytwarzania oprogramowania używanych przez organizacje twierdzące, iż używają metod zwinnych. Wspólne praktyki wykorzystywane przez większość organizacji zwinnych obejmują: wspólne tworzenie historyjek użytkownika (ang. collaborative user story creation), retrospektywy, ciągłą integrację oraz planowanie zarówno całych wydań jak i każdej iteracji. W tej sekcji, przejrzymy te techniki i praktyki.

1.2.1 Podejścia do zwinnego wytwarzania oprogramowania

Nie istnieje jedno podejście do zwinnego wytwarzania oprogramowania, ale wiele różnych podejść. Każde z nich inaczej implementuje wartości i zasady z Manifestu Agile. W niniejszym sylabusie, przyjrzymy się popularnym reprezentantom podejść zwinnych: programowaniu ekstremalnemu (ang. „eXtreme Programming"), Scrumowi oraz Kanbanowi.

Programowanie ekstremalne (ang. „eXtreme Programming")

Programowanie ekstremalne (ang. „eXtreme Programming", XP), pierwotnie wprowadzone przez Kenta Becka [Beck04], jest zwinnym podejściem do wytwarzania oprogramowania opisanym przez pewne wartości, zasady i praktyki wytwarzania.

XP obejmuje pięć wartości kierujących wytwarzaniem: komunikacja, prostota, informacje zwrotne, odwaga i szacunek.

XP opisuje zbiór zasad, jako dodatkowe wytyczne: humanitaryzm, ekonomia, wzajemny zysk, samopodobieństwo, doskonalenie, różnorodność, rozwaga, przepływ, okazja, redundancja, niepowodzenie, jakość, małe kroczki i zaakceptowana odpowiedzialność.

XP opisuje trzynaście podstawowych praktyk: dzielenie wspólnej przestrzeni, bogata w informacje przestrzeń pracy (ang. informative workspace), pobudzająca praca, programowanie w parach, historyjki, cykle tygodniowe, cykle kwartalne, luz, dziesięciominutowe budowanie wersji, ciągła integracja, programowanie w oparciu o zasadę „najpierw test" oraz podejście przyrostowe.

XP, jego wartości i zasady istotnie wpływają na większość zwinnych podejść do wytwarzania oprogramowania obecnie wykorzystywanych. Np. zespoły zwinne wykorzystujące Scrum często stosują praktyki z XP.

Scrum

Scrum, jest zwinną metodyką zarządczą, która zawiera następujące składowe narzędzia i praktyki [Schwaber01]:

- Sprint: Scrum dzieli projekt na iteracje o stałej długości (na ogół dwa do czterech tygodni).
- Przyrost produktu: Wynikiem każdego sprintu jest produkt, który potencjalnie można wdrożyć (nazywany przyrostem).
- Backlog produktu: Właściciel produktu zarządza spriorytetyzowaną listą planowanych pozycji do zrobienia (nazywaną backlogiem produktu). Zadania produktu zmieniają się od sprintu do sprintu (nazywamy to udoskonalaniem backlogu produktu).
- Backlog sprintu: na początku każdego sprintu zespół scrumowy wybiera z backlogu produktu zbiór pozycji o najwyższym priorytecie (nazywamy ten zbiór backlogiem sprintu). Ponieważ to zespół Scrumowy, a nie właściciel produktu, wybiera pozycje do realizacji podczas sprintu, wybór nawiązuje raczej do zasady pobierania niż do zasady wpychania pracy.
- Definicja ukończenia (ang. „Definition of Done"): By upewnić się, że na końcu sprintu otrzymamy produkt potencjalnie nadający się do wdrożenia, zespół Scrumowy przedyskutowuje i określa odpowiednie kryteria ukończenia sprintu. Dyskusja pogłębia

zrozumienie przez zespół poszczególnych pozycji z backlogu i wymagań dotyczących produktu.

- Ograniczenia czasowe (ang. „timeboxing"): Tylko te zadania, wymagania lub cechy, które zespół chce ukończyć w czasie sprintu, wchodzą do zadań sprintu. Jeżeli zadanie nie może być ukończone w czasie sprintu, odpowiednie cechy produktu są dzielone i zadanie wraca do backlogu produktu. Technika ta nazywa się ograniczaniem czasowym (ang. timeboxing), stosuje się ją nie tylko do zadań, ale również w innych sytuacjach (np. narzucanie czasów rozpoczynania i kończenia spotkań).
- Przejrzystość: Status sprintu jest raportowany i uaktualniany codziennie w czasie spotkań nazywanych codziennym Scrumem, dzięki czemu zawartość i postęp bieżącego sprintu – także wyniki testów – są dostępne dla zespołu Scrumowego, zarządzających i innych zainteresowanych. Np. status sprintu może być pokazywany na białej tablicy sucho-ścieralnej.

Scrum definiuje trzy role członków zespołu scrumowego:

- Scrum Master zapewnia, że praktyki i reguły scrumowe są implementowane i odpowiednio stosowane, rozwiązuje ich naruszenia, problemy ze zasobami i inne przeszkody, które mogłyby przeszkodzić zespołowi w stosowaniu reguł i praktyk. Scrum Master nie jest liderem zespołu, ale trenerem (coachem).
- Właściciel produktu reprezentuje użytkownika oraz tworzy, zarządza i nadaje priorytety w backlogu produktu. Właściciel produktu nie jest liderem zespołu.
- Zespół wytwórczy składa się z trzech do dziewięciu osób, które razem wytwarzają i testują produkt. Zespół sam się organizuje. Nie ma lidera zespołu, więc decyzje podejmowane są przez zespół. W zespole występuje wiele ról projektowych (patrz sekcje 2.3.2 oraz 3.1.4).

Scrum (w przeciwieństwie do XP) nie wymusza, które techniki wytwarzania oprogramowania (np. zasada „najpierw test") powinny być stosowane. Co więcej, Scrum nie dostarcza wytycznych, jak testowanie powinno być wykonywane w projekcie scrumowym.

Kanban

Kanban [Anderson13] jest podejściem zarządczym, często stosowanym w projektach zwinnych. Podstawowym jego celem jest wizualizacja i optymalizacja przepływu pracy w łańcuchu wartości dodanej (ang. a value-added chain). Kanban wykorzystuje trzy narzędzia [Linz14]:

- Tablicę kanbanową : Łańcuch wartości, którym zarządzamy, jest wizualizowany na tablicy kanbanowej. Każda kolumna pokazuje etap procesu - zbiór powiązanych czynności np. wytwarzanie, testowanie itp. Elementy, które powinny zostać wyprodukowane lub zadania, które będą przetwarzane, są symbolizowane przez kolorowe kartki. Kartki są przesuwane od lewej do prawej strony przez kolejne kolumny tablicy, które odpowiadają następującym po sobie etapom procesu.
- Limit pracy w toku (ang. Work in Progress): Ilość równolegle wykonywanych zadań jest ściśle ograniczona. Jest to kontrolowane przez ustalenie maksymalnej dozwolonej liczby kartek na danym etapie lub globalnie na tablicy. Gdy na danym etapie na tablicy pojawia się wolne miejsce pracownik przesuwa kartkę z wcześniejszego etapu.
- Czas realizacji (ang. lead time): Kanban jest używany do optymalizacji ciągłego przepływu zadań przez minimalizację (średniego) czasu realizacji dla całego łańcucha wartości.

Kanban wykazuje pewne podobieństwo do Scruma. W obu podejściach, wizualizacja aktywnych zadań (np. na dostępnej dla wszystkich tablicy ścieralnej) zapewnia przejrzystość zawartości i postępu zadań. Zadania jeszcze nie przydzielone czekają w zbiorze zadań do wykonania, w momencie, gdy zwalnia się miejsce w kolejnym etapie są one odpowiednio przesuwane na tablicy kanbanowej.

Zarówno iteracje jak i sprinty są opcjonalne w Kanbanie. Proces w Kanbanie pozwala na wydawanie produktów raczej jeden po drugim, niż jako część wydania. Ograniczenia czasowe (ang. timeboxing), jako mechanizm synchronizujący stają się tym samym opcjonalne, inaczej niż w Scrumie, gdzie synchronizuje się wszystkie zadania w ramach sprintu.

1.2.2 Wspólne tworzenie historyjek użytkownika

Słabej jakości specyfikacja – główny powód niepowodzenia projektu – może powstawać w wyniku braku zrozumienia przez użytkownika własnych potrzeb, braku globalnej wizji systemu, nadmiernych lub sprzecznych właściwości oraz innych błędów w przepływie informacji. W środowisku zwinnym, historyjki użytkownika pisane są, by uchwycić wymagania z perspektywy deweloperów, testerów i przedstawicieli biznesu. W przeciwieństwie do sekwencyjnego cyklu wytwarzania oprogramowania, gdzie wspólna wizja właściwości jest osiągana przez przeglądy formalne po utworzeniu wymagań, w zwinnym rozwoju oprogramowania wspólna wizja jest osiągana przez częste przeglądy nieformalne podczas pisania wymagań.

Historyjki użytkownika muszą obejmować zarówno właściwości funkcjonalne jak i niefunkcjonalne. Każda historyjka powinna zawierać kryteria akceptacji dla tych właściwości. Te kryteria powinny być definiowane we współpracy pomiędzy przedstawicielami biznesu, deweloperami i testerami. Kryteria te dostarczają deweloperem i testerom dokładniejszej wizji właściwości, którą reprezentanci biznesu będą sprawdzać. Zespół zwinny uważa zadanie za ukończone, gdy zbiór kryteriów akceptacyjnych jest spełniony.

Zazwyczaj unikalna perspektywa testera poprawia historyjkę użytkownika poprzez identyfikację brakujących szczegółów lub wymagań niefunkcjonalnych. Wkład testera może przyjmować formę zadawania przedstawicielom biznesu otwartych pytań dotyczących historyjki użytkownika, proponowania sposobów ich testowania oraz potwierdzania spełnienia kryteriów akceptacji.

Przy wspólnym tworzeniu historyjek użytkownika można wykorzystywać takie techniki jak burza mózgów i mapa myśli. Tester może też wykorzystywać technikę INVEST [INVEST]:

- **I**ndependent – niezależna
- **N**egotiable – negocjowalna
- **V**aluable – wartościowa
- **E**stimatable – dająca się oszacować
- **S**mall – niewielka
- **T**estable – testowalna

Zgodnie z koncepcją 3C [Jeffries01], historyjka użytkownika jest połączeniem trzech elementów:

- Karty (ang. Card): Karta jest fizycznym nośnikiem opisującym historyjkę. Identyfikuje wymaganie, jego krytyczność, oczekiwany czas wykonania i trwania testów oraz kryteria akceptacji dla tej historyjki. Opis powinien być precyzyjny, gdyż będzie używany w backlogu produktu.
- Konwersacja (ang. Conversation): Konwersacja wyjaśnia jak oprogramowanie będzie używane. Może to być udokumentowane lub przekazane ustnie. Testerzy, mający inny punkt widzenia niż deweloperzy czy przedstawiciele biznesu [ISTQB_FL_SYL], mogą wnieść wartościowy wkład w wymianę myśli, opinii i doświadczeń. Konwersacja zaczyna się podczas fazy planowania wydania i jest kontynuowana, gdy historyjka jest planowana.
- Potwierdzenie (ang. Confirmation): Kryteria akceptacji, omawiane podczas konwersacji, są używane do potwierdzenia, że historyjka jest ukończona. Te kryteria akceptacji mogą rozciągać się na wiele historyjek. Do sprawdzenia spełnienia kryteriów powinny zostać użyte testy zarówno pozytywne jak i negatywne. Podczas potwierdzania, różni uczestnicy grają rolę testera. Mogą to być deweloperzy jak i specjaliści od wydajności, bezpieczeństwa, współdziałania i innych cech jakościowych. By potwierdzić, że historyjka może zostać zamknięta, zdefiniowane kryteria akceptacji powinny zostać przetestowane i powinno być pokazane ich spełnienie.

Różne zespoły zwinne różnie dokumentują historyjki użytkownika. Niezależnie od podejścia, dokumentacja powinna być zwięzła, wystarczająca i ograniczać się do niezbędnych elementów.

1.2.3 Retrospektywy

W wytwarzaniu zwinnym, retrospektywa to spotkanie na końcu każdej iteracji, na którym dyskutuje się o tym, co się udało, co można poprawić, jak wdrożyć poprawki i powtórnie osiągnąć sukces w następnej iteracji. Retrospektywa obejmuje takie zagadnienia jak proces, ludzie, organizacja, relacje i narzędzia. Regularnie przeprowadzane spotkanie retrospektywne, po których następują odpowiednie działania, są krytyczne dla samoorganizacji i ciągłego doskonalenia wytwarzania oraz testowania.

Retrospektywa może owocować decyzjami dotyczącymi doskonalenia testów, zogniskowanymi na skuteczności testów, produktywności testów, jakości przypadków testowych oraz satysfakcji zespołu. Retrospektywy mogą również obejmować testowalność aplikacji, historyjek użytkownika, cech systemu oraz jego interfejsów. Ponadto analiza pierwotnych przyczyn defektów przeprowadzona podczas spotkania retrospektywnego może pomóc w usprawnieniu testowania oraz wytwarzania. Na ogół przyjmuje się, iż zespoły powinny wprowadzać tylko kilka usprawnień podczas jednej iteracji. Pozwala to na ciągłe doskonalenie w stałym tempie.

Czas i organizacja retrospektyw zależy od stosowanej metodyki zwinnej. Przedstawiciele biznesu oraz zespół są uczestnikami retrospektyw, a koordynator organizuje i przeprowadza spotkania. W pewnych sytuacjach zespół może zaprosić na spotkanie innych uczestników.

Testerzy powinni pełnić ważną rolę w retrospektywach. Testerzy są częścią zespołu i wnoszą swój unikalny punkt widzenia [ISTQB_FL_SYL, rozdz. 1.5]. Testowanie jest wykonywane w każdym sprincie i istotnie przyczynia się do sukcesu każdego sprintu. Wszyscy członkowie

zespołu, testerzy i nietesterzy, mogą wnosić wkład zarówno w czynności testowe jak i nietestowe.

Niezależnie od tego, kto bierze w nich udział, retrospektywy muszą odbywać się w profesjonalnym środowisku charakteryzującym się wzajemnym zaufaniem. Cechy udanych retrospektyw są takie same jak innych przeglądów [ISTQB_FL_SYL, sekcja3.2].

1.2.4 Ciągła integracja

Dostarczanie przyrostów produktu wymaga niezawodnego, pracującego, zintegrowanego oprogramowania na koniec każdego sprintu. Ciągła integracja podejmuje to wyzwanie poprzez regularne łączenie wszystkich zmian wykonanych w oprogramowaniu i integrację wszystkich zmienionych modułów przynajmniej raz dziennie. Zarządzanie konfiguracją, kompilacja, budowanie wersji, wprowadzanie i testowanie tworzą pojedynczy, zautomatyzowany, regularnie powtarzany proces. Ponieważ deweloperzy integrują swoją pracę stale, budują stale i testują stale, błędy w kodzie są wykrywane szybciej.

Proces ciągłej integracji składa się z następujących zautomatyzowanych czynności, następujących po kodowaniu, debagowaniu i wprowadzeniu kodu do wspólnego repozytorium:

- Statyczna analiza kodu: wykonanie i zaraportowanie wyników statycznej analizy kodu.
- Kompilacja: kompilowanie i integracja, generowanie kodu wykonywalnego.
- Testy jednostkowe: wykonanie testów, pomiar pokrycia kodu i raportowanie wyników tych testów.
- Wdrażanie: instalacja wersji w środowisku testowym.
- Testy integracyjne: wykonanie testów integracyjnych i zaraportowanie wyników.
- Raportowanie (tablica rozdzielcza): umieszczenie statusu wszystkich poprzednich czynności w publicznie dostępnym miejscu lub wysłanie statusu do zespołu.

Ten proces automatycznego budowania i testowania ma miejsce codziennie i wykrywa błędy integracji wcześnie i szybko. Ciągła integracja pozwala testerom zwinnym na wykonywanie testów automatycznych regularnie (w pewnych przypadkach, jako część samego procesu ciągłej integracji) i szybkie przekazywanie zespołowi informacji zwrotnej na temat jakości kodu. Te wyniki testów są dostępne dla każdego członka zespołu, szczególnie jeżeli zintegrowane z procesem zostanie automatyczne raportowanie. Automatyczne testy regresji mogą być wykonywane ciągle przez cały sprint. Dobre automatyczne testy regresji pokrywają tyle funkcjonalności, ile tylko jest możliwe, włącznie z historyjkami użytkownika dostarczonymi w poprzednich sprintach. Dobre pokrycie automatycznymi testami regresyjnymi wspiera budowanie dużych, zintegrowanych systemów. Gdy testy regresyjne są zautomatyzowane, testerzy zwinni mogą się skoncentrować na ręcznych testach nowych właściwości, implementowanych zmian oraz testach potwierdzających naprawienie defektów.

Jako uzupełnienie testów automatycznych, organizacje stosujące ciągłą integrację na ogół używają narzędzi do budowy wersji (ang. „build tools”) do wdrożenia ciągłej kontroli jakości. Poza wykonywaniem testów jednostkowych i integracyjnych, narzędzia te mogą wykonywać dodatkowo testy statyczne i dynamiczne, mierzyć i profilować wydajność, ekstrahować i formatować dokumentację z kodu źródłowego, a także ułatwiać manualne procesy zapewniania jakości. To nieustanne stosowanie kontroli jakości zmierza do podnoszenia jakości produktu jak

również zmniejszenia czasu dostarczenia jej przez zastąpienie tradycyjnej praktyki stosowania kontroli jakości po zakończeniu rozwoju całego produktu.

Narzędzia do budowy mogą być podłączone do narzędzi do automatycznego wdrażania (ang. automatic deployment tool), które mogą przenieść odpowiednią wersję z serwera do ciągłej integracji i budowy bezpośrednio do jednego lub kilku środowisk deweloperskich, testowych, pośrednich lub nawet produkcyjnych. Zmniejsza to liczbę błędów i opóźnień związanych z poleganiem na wyspecjalizowanej kadrze lub programistach instalujących wydania w tych środowiskach.
Ciągła integracja zapewnia następujące korzyści:

- Wcześniejsze wykrywanie i łatwiejszą analizę podstawowych przyczyn problemów integracyjnych oraz zmian powodujących konflikty.
- Regularną informację zwrotną niezależnie od tego czy kod działa.
- Testerzy zwinni są co najwyżej jeden dzień za deweloperską wersją oprogramowania.
- Redukcję ryzyka regresji związanego z refaktoryzacją kodu przez deweloperów.
- Zapewnienie, że codzienna praca deweloperów jest oparta na solidnych podstawach.
- Zapewnienie wizualizacji przyrostu produkcji, jednocześnie motywując deweloperów i testerów.
- Eliminację ryzyka harmonogramowego związanego z integracją „wielki wybuch”.
- Przez cały sprint stałą dostępność do wykonywalnego oprogramowania dla testów, celów demonstracyjnych lub szkoleniowych.
- Redukcję powtarzalnych testów manualnych.
- Zapewnienie szybkiej informacji zwrotnej dla decyzji o doskonaleniu jakości i testów.

Tym niemniej, ciągła integracja nie jest wolna od ryzyka i wyzwań:

- Należy wdrożyć i zarządzać narzędziami do ciągłej integracji.
- Proces integracji ciągłej musi być ustanowiony i stworzony.
- Automatyzacja testów wymaga dodatkowych zasobów i może być skomplikowana we wprowadzeniu.
- Wymagane jest dobre pokrycie testami, by móc skorzystać z dobrodziejstw automatyzacji testów.
- Zespoły czasami zbyt mocno polegają na testach jednostkowych, a wykonują zbyt mało testów systemowych i akceptacyjnych.

Ciągła integracja wymaga wykorzystania narzędzi, w tym narzędzi do testowania, narzędzi do automatyzacji procesu budowania oraz narzędzi do kontroli wersji.

1.2.5 Planowanie wydania i iteracji

Jak wspomniano w sylabusie poziomu podstawowego [ISTQB_FL_SYL] planowanie jest czynnością ciągłą i tak też jest w zwinnym cyklu życia. W zwinnym cyklu życia, mamy dwa rodzaje planowania: planowanie wydania i planowanie iteracji.

Planowanie wydania przewiduje wydanie produktu, często kilka miesięcy od rozpoczęcia projektu. Planowanie wydań definiuje i przedefiniowuje backlog produktu i może zawierać przerabianie dużych historyjek użytkownika w zbiór mniejszych historyjek. Planowanie wydania

dostarcza podstawy do podejścia do testów oraz do planu testów i rozciąga się na wszystkie iteracje. Plany wydania są wysokopoziomowe.

W planowaniu wydania, reprezentanci biznesu, we współpracy z zespołem (patrz sekcja 1.2.2), wybierają historyjki użytkownika i nadają im priorytety dla całego wydania. W oparciu o te historyjki, określane są ryzyka projektowe i jakościowe oraz wykonywane jest wysokopoziomowe szacowanie nakładu pracy (patrz sekcja 3.2).

Testerzy są zaangażowani w planowanie wydania będąc szczególnie użytecznymi przy następujących czynnościach:

- Definiowanie testowalnych historyjek użytkownika, włączając w to kryteria akceptacji.
- Branie udziału w analizie ryzyk projektowych i jakościowych.
- Szacowanie nakładu pracy związanego z historyjkami użytkownika.
- Definiowanie potrzebnych poziomów testów
- Planowanie testów dla wydania.

Gdy plan wydania jest gotowy, rozpoczyna się planowanie iteracji dla pierwszej iteracji. Przy planowaniu iteracji patrzy się tylko do końca iteracji, koncentrując się na backlogu sprintu.

Podczas planowania iteracji, zespół wybiera historyjki użytkownika z ustawionych według priorytetu w backlogu produktu, uszczegóławia historyjki użytkownika, wykonuje analizę ryzyka dla historyjek, oraz szacuje pracę potrzebną do wykonania każdej historyjki. Jeżeli jakaś historyjka użytkownika jest zbyt mglista i nie powiodą się próby jej rozjaśnienia, zespół może odrzucić tę historyjkę i wziąć następną historyjkę zgodnie z priorytetami. Reprezentanci biznesu mają obowiązek odpowiadać na pytania zespołu na temat każdej historyjki, tak by zespół wiedział jak implementować każdą historyjkę i jak ją testować.

Ilość wybranych historyjek zależy od oszacowanego tempa pracy zespołu (ang. „team velocity”) i wyszacowanego rozmiaru wybranych historyjek. Gdy zawartość iteracji zostanie uzgodniona, historyjki dzielone są na zadania, które zostaną wykonane przez odpowiednich członków zespołu.

Testerzy są zaangażowani w planowanie iteracji będąc szczególnie użytecznymi w następujących czynnościach:

- Udział w szczegółowej analizie ryzyka dla historyjek użytkownika.
- Określanie testowalności historyjek użytkownika.
- Tworzenie testów akceptacyjnych dla historyjek użytkownika.
- Dzielenie historyjek użytkownika na zadania (zwłaszcza zadania testowe).
- Szacowanie pracochłonności zadań testowych.
- Identyfikowanie funkcjonalnych i niefunkcjonalnych własności systemu, które trzeba przetestować.
- Wspieranie i udział w automatyzacji testów na różnych poziomach testowania.

Plany wydania mogą się zmieniać podczas trwania projektu, włączając w to zmiany w poszczególnych historyjkach użytkownika w backlogu produktu. Te zmiany mogą być uruchamiane przez czynniki wewnętrzne lub zewnętrzne. Czynniki wewnętrzne to możliwości

dostarczania, prędkość oraz czynniki techniczne. Czynniki zewnętrzne to odkrycie nowych rynków, nowi konkurenci lub wątki biznesowe, które mogą zmienić cele wydania lub terminy docelowe. Co więcej, plany iteracji mogą się zmieniać w trakcie iteracji. Na przykład, pewna historyjka użytkownika może okazać się bardziej złożona, niż założono podczas szacowania.

Tego typu zmiany mogą okazać się wyzwaniem dla testerów. Testerzy muszą rozumieć ogólny obraz wydania dla celów planowania testów. Dodatkowo muszą mieć adekwatną podstawę testów lub wyrocznię testową w każdej iteracji dla celów tworzenia testów [ISTQB_FL_SYL, sekcja 1.4]. Potrzebne informacje muszą być dostępne testerom dostatecznie wcześnie, a jednocześnie zmiany muszą być witane z otwartymi ramionami zgodnie z metodykami zwinnymi. Dylemat ten wymaga uważnych decyzji związanych ze strategiami testów i dokumentacją testów. Więcej na temat wyzwań zwinnych w [Black09, rozdział 12].

Planowanie wydań oraz iteracji powinno obejmować również planowanie testów, nie tylko planowanie zadań deweloperskich. Konkretne problemy związane z testami, którymi należy się zająć, obejmują:

- Zakres i rozmiar testów dla obszarów branych pod uwagę, cele testowania wraz z uzasadnieniem tych decyzji.
- Członków zespołów, którzy będą wykonywali zadania testowe.
- Potrzebne środowiska i dane testowe, na kiedy są one potrzebne oraz czy jakieś uzupełnienia lub zmiany w środowisku i danych testowych będą wymagane przed lub podczas projektu.
- Harmonogram, kolejność, zależności, wstępne wymagania dla zadań testowych związanych z testami funkcjonalnymi i niefunkcjonalnymi (np. jak często wykonywać testy regresywne, które cechy zależą od innych cech lub danych testowych itp.), włączając w to informacje jak zadania testowe zależą od zadań deweloperskich.
- Ryzyka projektowe i jakościowe do rozpatrzenia (patrz rozdz. 3.2.1).

Dodatkowo, dla większych zespołów, estymacja pracochłonności powinna brać pod uwagę czas i pracochłonność potrzebną do wykonania wymaganych zadań testowych.

2 Podstawowe zasady, praktyki i procesy w testowaniu zwinnym – 105 minut

Słowa kluczowe
element konfiguracji, test weryfikacji wersji, zarządzanie konfiguracją

Cele nauczania dla podstawowych zasad, praktyk i procesów w testowaniu zwinnym

2.1 Różnice pomiędzy tradycyjnym i zwinnym podejściem do testowania:

FA-2.1.1 (K2) Kandydat potrafi opisać różnice pomiędzy testowaniem w projektach zwinnych i tradycyjnych (nie zwinnych).

FA-2.1.2 (K2) Kandydat potrafi opisać, w jaki sposób czynności kodowania i testowania są zintegrowane w podejściu zwinnym.

FA-2.1.3 (K2) Kandydat potrafi opisać rolę niezależnego testowania w projektach zwinnych.

2.2 Status testowania w projektach zwinnych

FA-2.2.1 (K2) Kandydat potrafi opisać podstawowy zbiór produktów używanych do informowania o statusie testów w projekcie zwinnym, włączając w to postęp testów i jakość produktu.

FA-2.2.2 (K2) Kandydat potrafi opisać proces ewoluowania testów w czasie wielu iteracji i potrafi wyjaśnić, dlaczego automatyzacja testów jest istotna przy zarządzaniu ryzykiem regresji w projektach zwinnych.

2.3 Rola i umiejętności testera w zespole zwinnym

FA-2.3.1 (K2) Kandydat rozumie umiejętności (ludzkie, dziedzinowe i testowe) testera w projekcie zwinnym.

FA-2.3.2 (K2) Kandydat rozumie rolę testera w projekcie zwinnym.

2.1 Różnice pomiędzy tradycyjnym i zwinnym podejściem do testowania

Jak opisano w Sylabusie poziomu podstawowego [ISTQB_FL_SYL] i w [Black09], czynności testowe są powiązane z czynnościami kodowania, dlatego też, testowanie wygląda różnie w różnych cyklach życia oprogramowania. Testerzy muszą rozumieć różnice pomiędzy testowaniem w tradycyjnych modelach cyklu życia (np. sekwencyjnych jak model V, czy iteracyjnych jak RUP), a zwinnych cyklach życia (takich jak Scrum, czy programowanie ekstremalne XP), by pracować efektywnie i skutecznie. Modele zwinne różnią się w sposobie integrowania czynności testowych i deweloperskich, produktach projektowych, nazwach kryteriów wejściowych i wyjściowych używanych na różnych poziomach testowania oraz w sposobie, w jaki testowanie może być skutecznie wykorzystane.

Testerzy powinni pamiętać, że organizacje różnią się w istotny sposób w swych wdrożeniach cykli życia. Odchylenia od ideałów zwinnego cyklu życia (patrz sekcja 1.1) mogą stanowić rozsądne dostosowanie i adaptację praktyk. Zdolność adaptacji do kontekstu danego produktu, włącznie z faktycznie stosowanymi praktykami wytwarzania oprogramowania, jest kluczowym czynnikiem sukcesu testerów.

2.1.1 Czynności testowe i wytwórcze

Jedną z podstawowych różnic pomiędzy tradycyjnymi a zwinnymi cyklami życia jest idea bardzo krótkich iteracji, z których każda kończy się działającym oprogramowaniem dostarczającym interesariuszom biznesowym wartościowej funkcjonalności. Na początku projektu następuje okres planowania wydania. Następnie mamy ciąg iteracji. Na początku każdej iteracji występuje okres jej planowania. Po wypracowaniu zakresu iteracji, wybrane historyjki użytkownika są wytwarzane, integrowane z systemem oraz testowane. Tego typu iteracje są wysoce dynamiczne, z czynnościami wytwarzania, integracji i testowania trwającymi przez cały czas iteracji, z istotnym zrównolegleniem i zachodzeniem na siebie tych czynności. Czynności testowe wykonywane są przez cały czas (np. codziennie), a nie tylko jako końcowe czynności iteracji.

Testerzy, deweloperzy, interesariusze biznesowi mają swoje miejsce w testowaniu, podobnie jak w tradycyjnym cyklu życia. Deweloperzy wytwarzają testy jednostkowe podczas wytwarzania właściwości z historyjek użytkownika. Następnie testerzy testują te właściwości. Właściciele produktu także testują historyjki, gdy są one wytwarzane podczas iteracji. Właściciele produktu mogą wykorzystywać napisane przypadki testowe, ale mogą także eksperymentować z właściwościami i używać ich, by zapewnić szybką informację zwrotną dla deweloperów.

W pewnych przypadkach okresowo wykonywane są iteracje „scalające” i stabilizujące mające na celu rozwiązanie przewlekłych defektów lub innych form długu technicznego. Tym niemniej, najlepszą praktyką jest nieuznawanie właściwości za zakończonej dopóki nie została ona zintegrowana i przetestowana w systemie (patrz [Goucher09], rozdział 15). Inna dobra praktyka to rozwiązywanie usterek z poprzedniej iteracji na początku następnej iteracji, jako części zadań sprintu dla tej iteracji (nazywamy to „napraw najpierw usterki” (ang. „fix bugs first”)). Tym

niemniej niektórzy narzekają, że ta praktyka skutkuje sytuacjami, w których całość pracy, która ma być wykonana w danej iteracji nie jest znana i trudniej jest oszacować, kiedy pozostałe właściwości mogą być zakończone. Na końcu ciągu iteracji, może wystąpić zbiór czynności wydania, które przygotowują oprogramowanie do dostawy, chociaż w niektórych przypadkach dostawa następuje na końcu każdej iteracji.

Gdy jako jedną ze strategii testowych stosuje się testowanie w oparciu o ryzyko, podczas planowania wydania ma miejsce wysokopoziomowa analiza ryzyka. Testerzy często prowadzą tę analizę. Tym niemniej, konkretne ryzyka jakościowe związane z pojedynczą iteracją są identyfikowane i oceniane podczas planowania iteracji. Analiza ryzyka może wpływać zarówno na kolejność wytwarzania jak i na priorytet czy zakres testowania danej właściwości. Wpływa także na oszacowanie wymaganego nakładu pracy (np. liczby „punktów historyjki") dla każdej cechy.

W pewnych podejściach zwinnych (np. XP), wykorzystywana jest praca w parach. Praca w parach może dotyczyć dwójki testerów, wspólnie testujących właściwość oprogramowania. Praca w parach może także dotyczyć testera pracującego wspólnie z deweloperem przy wytwarzaniu i testowaniu właściwości. Praca w parach może być utrudniona, gdy zespół testowy jest rozproszony, jednakże można wdrożyć procesy i narzędzia umożliwiające rozproszoną prace w parach. Więcej na temat zagadnień dotyczących pracy zespołowej patrz [ISTQB_ALTM_SYL].

Testerzy mogą także pracować, w ramach zespołu, jako szkolący (coach) w sprawach dotyczących testowania i jakości, dzieląc się przy tym wiedzą związaną z testowaniem i wspierając prace związane z zapewnieniem jakości. Wspomaga to promocję idei wspólnej odpowiedzialności za jakość.

W wielu zespołach zwinnych ma miejsce automatyzacja testów na wszystkich poziomach. Może to oznaczać, że testerzy spędzają więcej czasu na tworzeniu, monitorowaniu oraz zarządzaniu testami automatycznymi i uzyskanymi wynikami. Z powodu intensywnego wykorzystania automatów testowych, coraz większa część testów manualnych w projektach zwinnych jest wykonywana z użyciem technik opartych na defektach oraz na doświadczeniu, takich jak ataki na oprogramowanie, testowanie eksploracyjne, czy też zgadywanie błędów (patrz [ISTQB_ALTA_SYL, sekcje 3.3 i 3.4] oraz [ISTQB_FL_SYL, sekcja 4.5]). Testerzy w projekcie zwinnym powinni być też gotowi do udziału w tworzeniu i wykonywaniu testów automatycznych. Ponieważ deweloperzy będą się skupić na tworzeniu testów jednostkowych, testerzy powinni się koncentrować na tworzeniu automatycznych testów integracyjnych, systemowych oraz całościowych opartych o scenariusze. Prowadzi to do tendencji faworyzowania w zespole zwinnym testerów o mocnych podstawach technicznych i umiejętnościach związanych z testami automatycznymi.

Jedną z podstawowych zasad zwinności jest to, że zmiany są dozwolone przez cały czas trwania projektu. Dlatego też, w zespołach zwinnych preferowana jest tzw. „lekka dokumentacja projektowa". Zmiana istniejącej właściwości ma wpływ na testy, szczególnie na testy regresji. Użycie testów automatycznych jest jedną z metod zarządzania nakładem pracy na testy związane za zmianą. Tym niemniej istotne jest, by zakres zmian nie przekroczył zdolności zespołu projektowego do radzenia sobie z ryzykami związanymi z tymi zmianami.

2.1.2 Produkty projektowe

Produkty projektowe dzielą się na trzy kategorie:

1. biznesowe produkty projektowe, opisujące co jest niezbędne (np. specyfikacja wymagań) i jak tego używać (np. dokumentacja użytkownika);
2. produkty wytwórcze, opisujące jak system jest zbudowany (np. diagram relacyjnej bazy danych), jak zaimplementowano system (np. kod) oraz jak sprawdzano poszczególne fragmenty kodu (np. automatyczne testy jednostkowe);
3. produkty testowe, które opisują jak system jest testowany (np. strategie i plany testów), które faktycznie testują system (np. testy automatyczne i manualne) oraz te, które przedstawiają wyniki testów (np. tablica rozdzielcza (ang. dashboard));

W typowej implementacji zwinnej, kładziony jest nacisk na ograniczenie pracochłonności związanej z tworzeniem i zarządzaniem dokumentacją, zgodnie z założeniem że bardziej wartościowe jest otrzymanie działającego oprogramowania, wraz z testami automatycznymi pokazującymi zgodność oprogramowania z wymaganiami. Zalecenie dotyczące redukcji dokumentacji stosuje się tylko do dokumentacji, która nie stanowi wartości dla odbiorcy. W udanej implementacji zwinnej, utrzymana jest równowaga pomiędzy – z jednej strony – rosnącym naciskiem na redukowanie dokumentacji oraz – z drugiej strony – dostarczaniem wystarczającej dokumentacji wspierającej czynności biznesowe, testowe, wytwórcze i zarządcze. Podczas fazy planowania wydania zespół musi podjąć decyzję, dotyczącą tego, które produkty są wymagane i jaki jest wymagany poziom dokumentacji.

Typowe biznesowe produkty w projekcie zwinnym, to historyjki użytkownika i kryteria akceptacji. Historyjki użytkownika w projekcie zwinnym są odpowiednikiem wymagań użytkownika. Powinny wyjaśniać, jak system powinien się zachowywać w odniesieniu do pojedynczej, spójnej właściwości lub funkcji. Historyjka użytkownika winna być na tyle mała, by można ją było zakończyć w jednej iteracji. Większe zbiory powiązanych cech lub zbiór pod-własności sumujących się w pojedynczą bardziej skomplikowaną cechę, są nazywane „epikami”. Epiki mogą zawierać historyjki użytkownika dla różnych zespołów wytwórczych. Na przykład, pojedyncza historyjka użytkownika może opisywać, co jest wymagane na poziomie API (warstwa pośrednicząca (ang. middleware)), a inna co jest potrzebne na poziomie interfejsu użutkownika (aplikacja). Zbiory te mogą być wytwarzane w serii sprintów. Każda z epik oraz zawarte w niej historyjki użytkownika mają zdefiniowane kryteria akceptacji, ustalające kiedy można uznać je za zakończone.

Typowe produkty deweloperskie w projektach zwinnych zawierają oczywiście kod. Tym niemniej, deweloperzy pracujący w projektach zwinnych często tworzą automatyczne testy jednostkowe. Testy te mogą powstawać po napisaniu kodu. Jednak w niektórych przypadkach programiści tworzą te testy przyrostowo przed napisaniem nowej porcji kodu, tak żeby stworzyć narzędzie do sprawdzenia czy nowo napisana porcja kodu działa zgodnie z oczekiwaniami.

Typowe produkty testowe w projektach zwinnych to testy automatyczne oraz takie dokumenty jak plan testów, katalog ryzyk jakościowych, testy manualne, raporty o defektach oraz logi wyników testów. Dokumenty te trzymane są w tak lekkiej postaci jak to tylko możliwe. Takie podejście jest również stosowane w tradycyjnym cyklu życia dla wielu tego typu dokumentów.

Należy dodać, że testerzy będą również wyliczali metryki testowe z raportów o usterkach i logów wyników testów. Tutaj również nacisk kładziony jest na lekkość podejścia.

W pewnych środowiskach zwinnych, zwłaszcza regulowanych (prawem), krytycznych ze względu na bezpieczeństwo, rozproszonych lub przy złożonych projektach czy produktach, wymagana jest dalsza formalizacja produktów projektowych. Na przykład, pewne zespoły przekształcają historyjki użytkownika i kryteria akceptacji w bardziej formalną specyfikację wymagań. Można przygotować raporty śledzenia pionowego i poziomego (ang. vertical and horizontal traceability reports), by spełnić wymagania audytorów, wymogi prawne oraz inne.

2.1.3 Poziomy testów

Poziomy testów to czynności testowe, logicznie powiązane, często przez dojrzałość lub kompletność testowanych elementów.

W sekwencyjnych modelach cyklu życia poziomy testów są często definiowane w ten sposób, że kryteria wyjścia z jednego poziomu są częścią kryteriów wejścia następnego poziomu. W pewnych modelach iteracyjnych ta reguła nie ma zastosowania. Poziomy testów się nakładają. Specyfikacja wymagań, specyfikacja projektowa oraz czynności wytwórcze zazębiają się z poziomami testów.

W niektórych zwinnych cyklach życia, zazębianie to może być spowodowane tym, iż zmiany w wymaganiach, projekcie czy też kodzie, mogą mieć miejsce w dowolnym momencie iteracji. O ile Scrum w teorii nie dopuszcza zmian w wymaganiach (tzn. w historyjkach użytkownika) po okresie planowania, to w praktyce można takie zmiany zaobserwować. Podczas iteracji, każda historyjka użytkownika będzie zwykle przechodziła kolejno przez następujące czynności testowe:

- testy jednostkowe, na ogół wykonywane przez deweloperów
- testy akceptacyjne właściwości, które czasami dzielone jest na dwie czynności:
    - testy weryfikacyjne właściwości, które są często automatyzowane; mogą być wykonywane przez deweloperów lub testerów i obejmują testowanie kryteriów akceptacyjnych historyjek użytkownika;
    - testy walidacyjne właściwości, które są zwykle manualne; mogą być wykonywane przez programistów, testerów i interesariuszy biznesowych pracujących razem by sprawdzić, czy dana cecha nadaje się do użycia, by lepiej pokazać postęp wykonanych prac oraz by otrzymać prawdziwą informację zwrotną od interesariuszy biznesowych.

Ponadto, podczas iteracji często występuje równoległy proces testów regresji. Obejmuje to ponowne uruchomienie automatycznych testów jednostkowych i testów weryfikacji właściwości z bieżącej i poprzednich iteracji, zwykle z wykorzystaniem narzędzi do ciągłej integracji (ang. continous integration framework).

W pewnych zespołach zwinnych istnieje także poziom testów systemowych, który rozpoczyna się, gdy tylko pierwsza historyjka użytkownika jest gotowa do takich testów. Ów poziom może obejmować zarówno testy funkcjonalne jak i niefunkcjonalne, tj. wydajność, niezawodność, użyteczność, instalowalność oraz inne istotne typy testów.

Zespoły zwinne mogą stosować różne formy testów akceptacyjnych (zgodnie z terminologią opisaną w sylabusie poziomu podstawowego). Wewnętrzne testy alfa oraz zewnętrzne testy beta mogą być przeprowadzane podczas zamykania każdej iteracji, po zamknięciu każdej iteracji lub po serii kilku iteracji.

2.1.4 Narzędzia do zarządzania testami i konfiguracją

Projekty zwinne często bardzo mocno wykorzystują narzędzia automatyzujące tworzenie, testowanie oraz zarządzanie tworzeniem oprogramowania. Deweloperzy wykorzystują narzędzia do analizy statycznej, testów jednostkowych i do mierzenia ich pokrycia. Programiści ciągle komitują kod i testy jednostkowe do systemów zarządzania konfiguracją przy użyciu struktur budowania i testowania. Takie struktury umożliwiają ciągłą integrację oprogramowania z systemem; analiza statyczna i testy jednostkowe są powtarzane, gdy tylko nowe oprogramowanie zostanie wprowadzane [Kubaczkowski].

Testy automatyczne mogą obejmować także testy funkcjonalne na poziomach integracyjnym i systemowym. Takie automatyczne testy funkcjonalne mogą być tworzone przy pomocy testowych jarzm funkcjonalnych, narzędzi o otwartym kodzie do funkcjonalnego testowania interfejsu użytkownika lub narzędzi komercyjnych i mogą być zintegrowane z testami automatycznymi, wykonywanymi jako część struktury do ciągłej integracji. W pewnych przypadkach, z powodu czasu potrzebnego do wykonania testów funkcjonalnych, mogą być one oddzielone od testów jednostkowych i wykonywane rzadziej. Na przykład, testy jednostkowe mogą być uruchamiane za każdym razem, gdy nowe oprogramowanie jest komitowane, podczas gdy długotrwałe testy funkcjonalne są uruchamiane tylko co kilka dni.

Jednym z celów testów automatycznych jest potwierdzenie, że testowana wersja działa i daje się zainstalować. Jeżeli którykolwiek test automatyczny nie został zaliczony, zespół powinien naprawić wykrytą usterkę do czasu następnego komitowania kodu. Wymaga to zainwestowania w raportowanie działające w czasie rzeczywistym, by móc zapewnić dobrą widoczność wyników testów. Tego typu podejście wspomaga redukcję kosztownych i nieefektywnych cykli „zbuduj–zainstaluj–awaria–przebuduj–ponownie zainstaluj”, często zdarzających się w wielu tradycyjnych projektach.

Inną korzyścią szerokiej automatyzacji testów i wykorzystania narzędzi do budowy jest to, że wspomaga ona zarządzanie ryzykiem regresji związanej z częstymi zmianami, które występują w projektach zwinnych. Jednakże, zbytnie poleganie jedynie na automatycznych testach jednostkowych w celu zarządzania tym ryzykiem może być problematyczne, bowiem testowanie jednostkowe ma ograniczoną skuteczność wykrywania defektów. [Jones11]. Automatyczne testy wymagane są także na poziomach integracji i systemowym.

2.1.5 Możliwości organizacyjne testowania niezależnego

Jak rozważano w sylabusie poziomu podstawowego [ISTQB_FL_SYL], niezależni testerzy są często bardziej efektywni w znajdowaniu defektów. W niektórych zespołach zwinnych, deweloperzy wytwarzają większość testów w formie testów automatycznych. Jeden lub więcej testerów może być członkiem zespołu, wykonując wiele zadań testowych. Jednakże

umieszczenie testera w zespole rodzi pewne ryzyko utraty jego niezależności i braku obiektywnej oceny.

Inne zespoły zwinne utrzymują w pełni niezależny, wydzielony zespól testerski i angażują testerów „na żądanie” na końcu każdego sprintu. Taki proces pozwala na zachowanie niezależności, co sprawia, że testerzy mogą dokonywać obiektywnej, bezstronnej oceny oprogramowania. Jednakże brak czasu, brak zrozumienia nowych właściwości produktu oraz problemy w relacjach z interesariuszami biznesowymi i deweloperami mogą często powodować kłopoty przy takim podejściu.

Trzecia opcja to posiadanie niezależnego, oddzielonego zespołu testowego, gdzie testerzy są przydzieleni do zespołu zwinnego na zasadzie długoterminowej na początku projektu, co umożliwia im utrzymanie niezależności z jednoczesnym poznaniem produktu i nawiązaniem mocnych relacji z innymi członkami zespołu. Dodatkowo, niezależny zespół testowy może utrzymywać kilku wyspecjalizowanych testerów (poza zespołem zwinnym) pracujących nad zadaniami długoterminowymi lub nienależącymi do sprintu, jak np. tworzenie narzędzi do automatycznego testowania, wykonywanie testów niefunkcjonalnych, tworzenie i utrzymywanie środowisk i danych testowych oraz zajmowanie się poziomami testów, które trudno wpasować w sprinty (np. testy integracji systemów).

2.2 Status testowania w projektach zwinnych

W projektach zwinnych zmiany pojawiają się nagle. Nagła zmiana oznacza, że status oraz postęp testów, jak i jakość produktu mogą ewaluować w sposób ciągły, a testerzy muszą znaleźć sposób na przekazywanie tych informacji zespołowi, tak by mógł podejmować trafne decyzje umożliwiające podążanie w kierunku skutecznego zakończenia każdej iteracji. Co więcej, zmiany mogą oddziaływać na istniejące już właściwości wytworzone w poprzednich iteracjach. Dlatego też, testy manualne i automatyczne musza być uaktualniane, aby radzić sobie z ryzykiem regresji.

2.2.1 Przekazywanie informacji o statusie testów, ich postępie oraz o jakości produktu

Zespoły zwinne posuwają się do przodu, dążąc do uzyskania działającego oprogramowania na końcu każdej iteracji. By określić, czy zespół będzie miał działające oprogramowanie, musi on monitorować postęp we wszystkich pracach w trakcie iteracji i wydania. Testerzy w zespołach zwinnych wykorzystują różne metody w celu rejestrowania postępu i statusu testów, m.in. wyniki testów automatycznych, postępu w zadaniach testowych i historyjkach, z tablicy zadań zwinnych oraz wykres spalania (ang. burndown charts) pokazujący postęp prac zespołu. Może to być komunikowane reszcie zespołu przy użyciu takich środków jak tablica rozdzielcza w wiki, maile w stylu tablicy rozdzielczej, a także werbalnie podczas codziennych spotkań porannych (ang. stand-up meeting). Zespoły zwinne mogą używać narzędzi, które automatycznie generują raporty o statusie w oparciu o wyniki testów i postęp prac, co z kolei uaktualnia tablicę rozdzielczą w wiki oraz maile. Ta metoda komunikacji zbiera także metryki z procesu testowego, co może być wykorzystane do doskonalenia procesu. Komunikowanie statusu procesu testowego w ten sposób zabiera mniej czasu testerom, którzy dzięki temu mogą się skoncentrować na projektowaniu i wykonywaniu większej liczby przypadków testowych.

Zespoły mogą używać wykresów spalania do śledzenia postępu prac przez cały okres wydania, a także podczas każdej z iteracji. Wykres spalania [Crispin09] przedstawia ilość pracy pozostałej do wykonania w stosunku do czasu przeznaczonego na wydanie lub iterację.

By zapewnić stałe, szczegółowe wizualne przedstawienie aktualnego statusu całego zespołu, włączając w to status testów, zespoły mogą wykorzystywać zwinną tablicę zadań (ang. Agile task board). Karty historyjek, zadania deweloperskie, zadania testowe i inne zadania powstałe podczas planowania iteracji (patrz sekcja 1.2.5) są prezentowane na tablicy zadań, często przy pomocy kolorowych kart, gdzie kolor określa typ zadania. Podczas iteracji, zarządzanie postępem prac odbywa się poprzez przesuwanie odpowiednich kart zadań na tablicy miedzy kolumnami takimi jak: „do zrobienia", „w toku", „weryfikacja" oraz „zrobione". Zespoły zwinne mogą używać narzędzi do obsługi kart historyjek oraz tablicy zadań, które to narzędzia mogą automatyzować uaktualnianie tablicy rozdzielczej i statusu.

Zadania testowe na tablicy zadań są powiązane z kryteriami akceptacyjnymi zdefiniowanymi dla każdej historyjki użytkownika. Gdy skrypty testów automatycznych, testy manualne i eksploracyjne dla zadania testowego osiągną status „wykonane", zadanie jest przenoszone do kolumny „zrobione" na tablicy zadań. Cały zespół regularnie przegląda status zadań na tablicy, często podczas codziennych Scrumów, by mieć pewność, że zadania są przemieszczane na tablicy w akceptowalnym tempie. Jeżeli jakieś zadanie (także testowe) nie przesuwa się lub przesuwa się zbyt wolno, jest ono specjalnie oznaczane, by zespół mógł je przejrzeć i rozwiązać kwestie blokujące postęp zadania.

Codzienny Scrum angażuje wszystkich członków zespołu, także testerów. Na tym spotkaniu podają oni aktualny status swoich zadań. Agenda dla każdego członka zespołu to [Agile Alliance Guide]:

- Co skończyłeś od poprzedniego spotkania?
- Co planujesz skończyć do następnego spotkania?
- Co blokuje ci prace?

Wszystkie problemy, które mogą blokować postęp prac, są przedstawiane podczas codziennych spotkań, więc cały zespół jest ich świadomy i rozwiązuje je razem.

Wiele zespołów zwinnych robi sondaże satysfakcji klienta, by poprawić jakość całego produktu i otrzymać informacje zwrotne na ile produkt spełnia oczekiwania klienta. W celu poprawy jakości zespoły mogą używać metryk podobnych do tych wykorzystywanych w tradycyjnych metodykach wytwarzania takich jak wskaźnik „zaliczone/niezaliczone", wskaźnik wykrywania awarii, wyniki testów potwierdzających i regresywnych, gęstość defektów, usterki znalezione i naprawione, pokrycie wymagań, pokrycie ryzyka, pokrycie kodu oraz modyfikacje kodu. Jak w każdym cyklu życia to, co metryki ujmują i raportują powinno być odpowiednie i ułatwiające podejmowanie decyzji. Metryki nie powinny być używane do nagradzania, karania lub separowania członków zespołu.

2.2.2 Zarządzanie ryzykiem regresji przy pomocy zmieniających się manualnych i zautomatyzowanych przypadków testowych

W projektach zwinnych, produkt rośnie po zakończeniu każdej iteracji. Tym samym zwiększa się zakres testów. Wraz z testami zmian kodu dokonanych w czasie bieżącej iteracji, testerzy sprawdzają także, czy nie obniżono jakości właściwości, które były wytworzone i przetestowane w poprzednich iteracjach. Ryzyko regresu właściwości jest wysokie w wytwarzaniu zwinnym, ze względu na rozległe zmiany w kodzie (dodawanie, modyfikowanie i usuwanie linii kodu z wersji na wersję). Ponieważ reagowanie na zmiany jest podstawową zasadą zwinności, dlatego też, w celu zaspokojenia potrzeby biznesowej, zmianom mogą podlegać uprzednio dostarczone właściwości. By utrzymać prędkość wytwarzania bez zaciągania dużego długu technicznego, krytyczna staje się jak najwcześniejsza inwestycja zespołu w automatyzację testów na wszystkich poziomach testów. Krytyczne jest także, by wszystkie produkty testowe takie jak: testy automatyczne, manualne przypadki testowe, dane testowe i inne artefakty testowe były aktualizowane w każdej iteracji. Dlatego też zaleca się, by wszystkie produkty testowe były zarządzane przy użyciu narzędzia do zarządzania konfiguracją. Zapewnia to kontrolę wersji, ułatwia dostęp wszystkim członkom zespołu, ułatwia zmiany wymagane przez zmieniającą się funkcjonalność, a równocześnie ciągle zachowuje historyczne informacje o produktach testowych.

Ponieważ kompletne powtarzanie wszystkich testów jest rzadko możliwe, zwłaszcza w napiętym czasowo projekcie zwinnym, testerzy w każdej iteracji powinni przeznaczyć czas na przejrzenie manualnych i zautomatyzowanych przypadków testowych z poprzednich i z aktualnej iteracji, by wybrać przypadki testowe, które mogą być kandydatami do zestawu testów regresji i opuścić te, które już nie są odpowiednie. Testy napisane w poprzednich iteracjach, sprawdzające konkretne właściwości, mogą mieć małą wartość w późniejszych iteracjach, ze względu na zmiany we właściwościach lub nowe właściwości zmieniające sposób zachowania się właściwości wcześniejszych.

Podczas przeglądania przypadków testowych, testerzy winni rozważać, czy przypadek nadaje się do automatyzacji. Zespół powinien automatyzować tyle przypadków testowych (z poprzednich i bieżącej iteracji), ile tylko jest to możliwe. Pozwala to na ograniczenie ryzyka regresji przy mniejszym nakładzie pracy niż wymagałoby testowanie regresji manualnie. Zmniejszony nakład pracy pozwala testerom na dokładniejsze testy nowych właściwości i funkcji w bieżącej iteracji.

Konieczne jest, by testerzy mieli umiejętności do szybkiej identyfikacji i poprawy przypadków testowych z poprzednich iteracji lub wydań, na które mają wpływ zmiany wykonane w bieżącej iteracji. Podczas planowania wydania powinno zostać określone, jak zespół projektuje, pisze i przechowuje przypadki testowe. Dobre praktyki projektowania testów i ich implementacji powinny być wcześnie wdrażane i konsekwentnie stosowane. Krótszy czas testowania oraz ciągłe zmiany wzmocnią wpływ złych praktyk projektowania i implementacji testów.

Użycie automatyzacji testów pozwala zespołom zwinnym na szybkie dostarczanie informacji zwrotnej o jakości produktu. Dobrze napisane testy automatyczne stanowią żyjącą dokumentację funkcjonalności systemu [Crispin09]. Umieszczając testy automatyczne i ich wyniki w systemie do zarządzania konfiguracją, razem z powiązanym ze źródłową wersją

produktu, zespoły zwinne mogą przeglądać przetestowane funkcjonalności i wyniki testów dla dowolnej wersji w dowolnym momencie czasu.

Automatyczne testy jednostkowe są wykonywane, zanim kod źródłowy zostanie umieszczony w głównym strumieniu kodu w systemie do zarządzania konfiguracją, po to aby zapewnić, że zmiany w kodzie nie zepsują budowanej wersji. By zredukować możliwość popsucia wersji, co może powodować spowolnienie postępu pracy całego zespołu, kod nie powinien być komitowany dopóki nie przejdą wszystkie automatyczne testy jednostkowe. Wyniki zautomatyzowanych testów jednostkowych zapewniają natychmiastową informację zwrotną dotyczącą jakości kodu i wersji, ale nie jakości produktu.

Zautomatyzowane testy akceptacyjne są wykonywane regularnie jako część ciągłej integracji wersji całego systemu. Testy te są wykonywane dla całej wersji systemu co najmniej raz dziennie, ale generalnie nie są wykonywane z każdym komitowaniem kodu, bo trwają one dłużej niż zautomatyzowane testy jednostkowe, co może spowodować spowolnienie komitowania kodu. Wyniki automatycznych testów akceptacyjnych zapewniają informację zwrotną o jakości produktu uwzględniając regresję w stosunku do ostatniej wersji, ale nie zapewniają informacji o całkowitej jakości produktu.

Zautomatyzowane testy systemu powinny być wykonywane bez przerwy. Początkowy podzbiór testów automatycznych pokrywający krytyczną funkcjonalność systemu i punkty integracji, powinien być wytworzony natychmiast po tym, jak nowa wersja jest instalowana w środowisku testowym. Testy te nazywamy testami weryfikacji wersji. Wyniki z testów weryfikacji wersji powinny zapewnić stałą informację zwrotną o podstawowej gotowości oprogramowania po instalacji, w ten sposób zespół nie inwestuje zbyt dużo czasu na testowanie wersji niestabilnej.

Testy automatyczne umieszczone w zbiorze testów regresji są zazwyczaj wykonywane jako część codziennego, głównego budowania w środowisku ciągłej integracji, a także za każdym razem, gdy nowa wersja jest umieszczana w środowisku testowym. Jeżeli zautomatyzowane testy regresji nie zostaną zaliczone, zespół zatrzymuje się i sprawdza przyczyny niezaliczenia testu. Test może nie zostać zaliczony ze względu na uzasadnione zmiany funkcjonalności w bieżącej iteracji; w tym przypadku test lub historyjka użytkownika powinny być uaktualnione, tak by odzwierciedlać nowe kryterium akceptacji. Test może zostać usunięty, jeżeli stworzony został nowy test pokrywający zmiany. Jednak, gdy test nie przeszedł z powodu usterki, dobra praktyka zespołu nakazuje naprawić usterkę, nim będzie kontynuowana praca nad nowymi cechami.

Poza automatyzacją testów następujące zadania mogą zostać zautomatyzowane:

- generacja danych testowych;
- wgrywanie danych testowych do systemu;
- umieszczanie wersji w środowisku testowym;
- przywracanie środowiska testowego (np. bazy danych lub plików witryny internetowej) do konfiguracji podstawowej;
- porównywanie danych wynikowych.

Automatyzacja tych zadań zmniejsza obłożenie zespołu i pozwala zespołowi na poświęcenie większej ilości czasu na rozwój i testy nowych cech.

2.3 Rola i umiejętności testera w projekcie zwinnym

W zespole zwinnym, testerzy muszą blisko współpracować z pozostałymi członkami zespołu i interesariuszami biznesowymi. Ma to wpływ na umiejętności, które tester powinien posiadać i czynności, jakie wykonuje w zespole zwinnym.

2.3.1 Umiejętności testera zwinnego

Testerzy w zespole zwinnym powinni posiadać wszystkie umiejętności, o których mowa w sylabusie poziomu podstawowego. Prócz tych umiejętności, tester w zespole zwinnym bardzo skorzysta na biegłości w automatyzacji testów, w wytwarzaniu sterowanym testami, w wytwarzaniu sterowanym testami akceptacyjnymi i w testowaniu białoskrzynkowym.

Ponieważ metodyki zwinne są silnie oparte na współpracy, komunikacji i interakcjach zarówno pomiędzy członkami zespołu jak i interesariuszami spoza zespołu, testerzy w zespołach zwinnych powinni być świadomi faktu, że istotne są też ich umiejętności interpersonalne.

Testerzy w zespołach zwinnych powinni:

- Być pozytywni i zorientowani na rozwiązywanie problemów we współpracy z członkami zespołu i interesariuszami.
- Wykazywać krytyczne, zorientowane na jakość, sceptyczne myślenie o produkcie.
- Aktywnie zdobywać informacje od interesariuszy (nie polegać wyłącznie na spisanej specyfikacji).
- Dokładnie sprawdzać i raportować wyniki testów, postęp testów i jakość produktu.
- Skutecznie pracować z przedstawicielami klienta i interesariuszami definiując testowalne historyjki użytkownika (przede wszystkim kryteria akceptacyjne).
- Współpracować w zespole, pracując w parach z programistami lub innymi członkami zespołu.
- Szybko reagować na zmiany, przez np. zmienianie, dodawanie i poprawianie przypadków testowych.
- Planować i organizować swoją własną pracę.

Stały rozwój, w tym także wzrost umiejętności interpersonalnych, jest niezbędny dla wszystkich testerów, także dla testerów zwinnych.

2.3.2 Rola testera w zespole zwinnym

Rola testera w zespole zwinnym obejmuje czynności, które generują i dostarczają informacje zwrotne dotyczące nie tylko statusu testów i jakości produktu, ale także jakości procesu. Poza czynnościami opisanymi w innych miejscach tego sylabusa, czynności te obejmują:

- Zrozumienie, implementowanie i poprawianie strategii testów.
- Mierzenie i raportowanie pokrycia testowego względem wszystkich dostępnych wymiarów pokrycia.
- Zapewnienie poprawnego użycia narzędzi testowych.
- Konfigurowanie, używanie i zarządzanie środowiskiem testowym i danymi testowymi.
- Raportowanie defektów i współpraca z zespołem przy ich rozwiazywaniu.

- Trenowanie (coaching) innych członków zespołu w różnych aspektach związanych z testowaniem.
- Zapewnienie, że odpowiednie zadania testowe są harmonogramowane podczas planowania wydania i iteracji.
- Aktywna współpraca z deweloperami, interesariuszami biznesowymi i właścicielami produktu, by poprawnie interpretować wymagania, zwłaszcza w zakresie ich testowalności, spójności i kompletności.
- Proaktywne uczestnictwo w retrospektywach zespołu, sugerowanie i implementowanie udoskonaleń.

W zespole zwinnym każdy członek zespołu jest odpowiedzialny za jakość produktu i ma swoją rolę w spełnianiu zadań związanych z testami.

Organizacje zwinne mogą natrafiać na pewne ryzyka organizacyjne związane z testowaniem:

- Testerzy tak blisko pracują z deweloperami, że tracą odpowiednie, testerskie, nastawienie.
- Testerzy stają się tolerancyjni bądź przemilczają nieskuteczne, nieefektywne lub niskojakościowe praktyki w zespole.
- Testerzy nie dają sobie rady z nadchodzącymi zmianami w iteracjach ograniczonych czasowo.

By ograniczyć te ryzyka, organizacje mogą rozważać różne odmiany metod zachowania niezależności, dyskutowane w sekcji 2.1.5.

3. Metody, techniki i narzędzia w testowaniu zwinnym - 480 minut

Słowa kluczowe

automatyzacja wykonania testu, karta testów, podejście do testów, ryzyko jakościowe, ryzyko produktowe
strategia testów, struktura do testów jednostkowych, szacowanie testów, taksonomia defektów, testowanie akceptacyjne, testowanie eksploracyjne, testowanie pielęgnowalności, testowanie regresywne, testowanie użyteczności, testowanie wydajnościowe, testowanie zabezpieczeń, wytwarzanie sterowane testami (TDD)

Cele nauczania dla metod, technik i narzędzi w testowaniu zwinnym

3.1 Metody testowania zwinnego

FA-3.1.1 (K1) Kandydat pamięta pojęcia: wytwarzanie sterowane testami (TDD), wytwarzanie sterowane testami akceptacyjnymi (ATDD), wytwarzanie sterowane zachowaniem (BDD).

FA-3.1.2 (K1) Kandydat pamięta pojęcie piramidy testowa.

FA-3.1.3 (K2) Kandydat potrafi opisać kwadranty testowe i ich związki z poziomami testów i typami testów.

FA-3.1.4 (K3) Kandydat potrafi dla danego projektu zwinnego, pełnić rolę testera w zespole scrumowym.

3.2 Ocena ryzyk jakościowych i szacowanie wysiłku testowego

FA-3.2.1 (K3) Kandydat potrafi ocenić ryzyka jakościowe produktu w projekcie zwinnym.

FA-3.2.2 (K3) Kandydat potrafi oszacować nakład pracy testowej na podstawie zawartości iteracji i ryzyk jakościowych produktu.

3.3 Techniki w projektach zwinnym

FA-3.3.1 (K3) Kandydat potrafi zinterpretować odpowiednie informacje wspomagające czynności testowe.

FA-3.3.2 (K2) Kandydat potrafi wyjaśnić interesariuszom biznesowym jak definiować testowalne kryteria akceptacyjne.

FA-3.3.3 (K3) Mając daną historyjkę użytkownika, kandydat potrafi napisać przypadki testowe dla wytwarzania sterowanego testami akceptacyjnymi.

FA-3.3.4 (K3) Mając daną historyjkę użytkownika, kandydat potrafi napisać przypadki testowe, zarówno funkcjonalne jak i niefunkcjonalne używając technik czarnoskrzynkowych.

FA-3.3.5 (K3) Kandydat potrafi wykonać testy eksploracyjne, by wspomóc testowanie w projekcie zwinnym.

3.4 Narzędzia w projektach zwinnych

FA-3.4.1 (K1) Kandydat zna różne narzędzia dostępne testerom oraz ich podział według przeznaczenia i czynnościami w projekcie zwinnym.

3.1 Metody testowania zwinnego

Niektóre praktyki testowania można wykorzystywać w dowolnym projekcie rozwojowym (zwinnym i tradycyjnym) w celu wytworzenia produktów o wysokiej jakości. Zaliczają się do nich pisanie testów z wyprzedzeniem, by wyrazić poprawne zachowanie, koncentrowanie się na wczesnym zapobieganiu, wykrywaniu i usuwaniu usterek oraz zapewnienie, że odpowiedni typ testów będzie wykonany w odpowiednim czasie na odpowiednim poziomie testów. Praktycy metodyk zwinnych dążą do wdrożenia tych praktyk jak najwcześniej. Testerzy w projektach zwinnych pełnią kluczową rolę w przewodzeniu w wykorzystaniu tych praktyk testowych w przez cały cykl życia oprogramowania.

3.1.1 Wytwarzanie sterowane testami (TDD), wytwarzanie sterowane testami akceptacyjnymi (ATDD) oraz wytwarzanie sterowane zachowaniem (BDD)

TDD, ATDD oraz BDD to trzy uzupełniające się nawzajem techniki, które są wykorzystywane przez zwinne zespoły przy wykonywaniu testów na różnych poziomach. Każda z tych technik jest wyrazem jednej z podstawowych zasad testowania, że korzystne dla projektu jest jak najwcześniejsze testowanie i zapewnienie jakości, ponieważ w technikach tych testy są definiowane przed kodem.

Wytwarzanie sterowane testami

Wytwarzanie sterowane testami to technika wytwarzania kodu kierowana przez zautomatyzowane przypadki testowe. Proces wytwarzania sterowanego testami zawiera:

- Dodanie testu, który stanowi wyobrażenie programisty o pożądanej funkcjonalności małego fragmentu kodu.
- Wykonanie testu, który nie powinien przejść, bo kod jeszcze nie istnieje.
- Naprzemienne pisanie kodu i uruchamianie testu, aż test przejdzie.
- Refaktoryzacja kodu po tym jak test przeszedł, ponowne uruchomienie testu w celu upewnienia się, że nadal przechodzi po tym jak kod został zrefaktoryzowany.
- Powtarzanie procesu dla następnego małego fragment kodu z uruchamianiem zarówno poprzednich testów, jak i nowo dodanych.

Stworzone przypadki testowe są przede wszystkim testami jednostkowymi i są zwykle zorientowane na kod (białoskrzyskowe). Jednakże testy mogą być też tworzone na poziomie integracyjnym lub systemowym. Wytwarzanie sterowane testami zostało spopularyzowane przez Programowanie Ekstremalne (XP) [Beck02], ale jest z powodzeniem używane w innych metodykach zwinnych, a czasem także w sekwencyjnym cyklu życia. Podejście to pomaga programistom skupić się na jasno zdefiniowanych wynikach oczekiwanych. Testy te są zautomatyzowane i używane przy ciągłej integracji.

Wytwarzanie sterowane testami akceptacyjnymi

Wytwarzanie sterowane testami akceptacyjnymi [Adzic09] definiuje kryteria akceptacji oraz testy podczas tworzenia historyjek użytkownika (patrz podrozdział 1.2.2). ATDD jest podejściem opartym na współpracy, które pozwala każdemu z interesariuszy na zrozumienie, jak każdy z

modułów ma się zachowywać oraz co mają zrobić programiści, testerzy oraz reprezentanci biznesu, żeby zapewnić dany sposób zachowania. Proces wytwarzania sterowanego testami akceptacyjnymi został wyjaśniony w podrozdziale 3.2.2.

ATDD tworzy reużywalne testy do wykorzystania w testach regresyjnych. Tworzenie i wykonywanie takich testów jest wspierane przez specyficzne narzędzia, często w procesie ciągłej integracji. Narzędzia te potrafią połączyć się z warstwami danych i usług, co pozwala na wykonywanie testów systemowych i akceptacyjnych. Wytwarzanie sterowane testami akceptacyjnymi pozwala na szybkie naprawienie defektów oraz walidację zachowania właściwości. Pomaga ono stwierdzić, czy kryteria akceptacyjne zostały spełnione.

Wytwarzanie sterowane zachowaniem (BDD)

Wytwarzanie sterowane zachowaniem [Chelimsky10] pozwala programiście na skupienie się testowaniu kodu w oparciu o wymagane zachowanie oprogramowania. Ponieważ testy oparte są o faktyczne zachowanie oprogramowania, są one łatwiejsze do zrozumienia przez członków zespołu oraz interesariuszy.

Przy definiowaniu kryteriów akceptacji w oparciu o format “mając/kiedy/wtedy” mogą być używane konkretne struktury (ang. framework) do wytwarzania sterowanego zachowaniem:

Mając pewien kontekst początkowy,
Kiedy nastąpi zdarzenie,
Wtedy zapewnione jest pewne wyjście.

3.1.2 Piramida testowa

System software’owy może być testowany na różnych poziomach. Typowe poziomy testów to, od najniższego do najwyższego, testy jednostkowe, integracyjne, systemowe i akceptacyjne (patrz [ISTQB FL_SYL], sekcja 2.2). Piramida testowa akcentuje potrzebę posiadania dużej liczby testów na niskim poziomie (dół piramidy). Gdy wytwarzanie przechodzi do wyższych poziomów, liczba testów maleje (szczyt piramidy). Na ogół testy jednostkowe i integracyjne są automatyzowane i tworzone przy pomocy narzędzi wykorzystujących interfejs programowania aplikacji (API). Na poziomie testów systemowych i akceptacyjnych, testy automatyczne są tworzone z wykorzystaniem narzędzi opartych o interfejs użytkownika. Pojęcie piramidy testowej jest oparte na testowej zasadzie wczesnego zapewnienia jakości i testowania, tzn. eliminacji usterek na jak najwcześniejszym etapie cyklu życia oprogramowania.

3.1.3 Kwadranty testowe, poziomy testów i typy testów

Kwadranty testowe, zdefiniowane przez Briana Maricka [Crispin09], przyporządkowują poziomowi testów odpowiedni typ testów w metodach zwinnych. Kwadranty testowe i ich modyfikacje pomagają w uwzględnieniu wszystkich ważnych typów testów i poziomów testów w cyklu życia wytwarzania. Ten model pokazuje też sposób na rozróżnienie i opisanie typów testów wszystkim interesariuszom, włącznie z deweloperami, testerami i przedstawicielami biznesu.

W modelu kwadrantów testowych, testy mogą być zorientowane na biznes (użytkownik) lub technologię (programista). Pewne testy wspomagają prace wykonywane przez zespół zwinny i potwierdzają zachowanie oprogramowania, inne z kolei weryfikują sam produkt. Testy mogą być w pełni manualne, w pełni zautomatyzowane, kombinacją testów manualnych i automatycznych, lub testami manualnymi wspomaganymi przez narzędzia. Istnieją następujące cztery kwadranty:

- Kwadrant Q1 jest na poziomie jednostkowym, zorientowany na technologię i wspomaga deweloperów. W tym kwadrancie znajdują się testy jednostkowe. Testy te powinny być zautomatyzowane i umieszczone w procesie ciągłej integracji.
- Kwadrant Q2 jest na poziomie systemowym, zorientowany na biznes i potwierdza zachowania się produktu. W tym kwadrancie zawarte są testy funkcjonalne, przykłady, testowanie historyjek, prototypy oparte na doświadczeniu użytkowników oraz symulacje. Te testy sprawdzają kryteria akceptacyjne. Mogą one być manualne lub zautomatyzowane. Często są tworzone podczas wytwarzania historyjek użytkownika i dlatego też poprawiają jakość historyjek. Są użyteczne do tworzenia zestawów automatycznych testów regresji.
- Kwadrant Q3 jest na poziomie systemowym lub akceptacji użytkownika, zorientowany na biznes i zawiera testy, które krytykują produkt używając realistycznych scenariuszy i danych. Ten kwadrant zawiera testy eksploracyjne, scenariusze, sprinty procesów, testowanie użyteczności, testy akceptacyjne użytkownika, testy alfa i beta. Te testy są często manualne i zorientowane na użytkownika.
- Kwadrant Q4 jest na poziomie systemowym lub akceptacji operacyjnej, zorientowany na technologię i zawiera testy, które krytykują produkt. Ten kwadrant zawiera testy wydajnościowe, obciążeniowe, przeciążające, testowanie skalowalności, zabezpieczeń, pielęgnowalności, zarządzani pamięcią, testowanie kompatybilności i współdziałania, migracji danych, infrastruktury oraz testowanie odzyskiwania. Testy te są często automatyzowane.

Podczas każdej iteracji, mogą być potrzebne testy z pojedynczego lub wszystkich kwadrantów. Kwadranty testowe mają zastosowanie raczej do testów dynamicznych niż statycznych.

3.1.4 Rola testera

Przez cały ten sylabus, odnosimy się głównie do metod i technik zwinnych, oraz omawiamy rolę testera w różnych zwinnych cyklach życia. W tej sekcji, przyjrzymy się roli testerów w projekcie zgodnym z cyklem życia według metodyki Scrum.

Praca zespołowa

Praca zespołowa to podstawowa zasada w wytwarzaniu zwinnym. Zwinność kładzie nacisk na zaangażowanie całego zespołu, składającego się z deweloperów, testerów i przedstawicieli biznesu, pracujących razem. Poniżej podano najlepsze praktyki organizacyjne i behawioralne w zespołach scrumowych:

- Interdyscyplinarny: Zespół pracuje sprawnie razem nad strategią testów, planowaniem testów, specyfikacją testów, wykonywaniem testów, oceną wyników testów i raportowaniem wyników z testów.

- Samoorganizujący się: Zespół może składać się z samych deweloperów, ale – jak podano w sekcji 2.1.5 – w sytuacjach idealnych do zespołu powinien dołączyć jeden lub więcej testerów.
- W jednym miejscu: Testerzy siedzą razem z deweloperami i właścicielem produktu.
- Współpracujący: Testerzy współpracują z innymi członkami zespołu, innymi zespołami, interesariuszami, właścicielem produktu i Scrum Masterem.
- Pełnomocny: Decyzje techniczne dotyczące projektowania i testowania są podejmowane przez członków zespołu jako całość (programiści, testerzy i Scrum Master), we współpracy z właścicielem produktu i z innymi zespołami w razie potrzeby.
- Zaangażowany: Tester jest zobowiązany do kwestionowania i oceniania zachowania oraz właściwości produktu, z uwzględnieniem oczekiwań i potrzeb klientów oraz użytkowników.
- Przejrzysty: Postęp prac programistycznych i testerskich jest widoczny na zwinnej tablicy zadań (patrz sekcja 2.2.1).
- Wiarygodny: Tester musi zapewnić wiarygodność strategii testowej, jej wdrożenia i wykonania. W przeciwnym wypadku interesariusze nie będą mieli zaufania do wyników testów. Często wiarygodność testów osiąga się przez dostarczanie interesariuszom informacji na temat procesu testowego.
- Otwarty na informację zwrotną: Informacja zwrotna jest istotnym aspektem osiągnięcia sukcesu w projekcie, zwłaszcza w projekcie zwinnym. Retrospektywy pozwalają zespołowi na uczenie się na sukcesach i porażkach.
- Elastyczny: Testowanie musi umożliwiać reakcję na zmiany, podobnie jak inne czynności w projektach zwinnych

Te najlepsze praktyki maksymalizują prawdopodobieństwo sukcesu testowania w projektach scrumowych.

Iteracja Zerowa

Iteracja zerowa jest pierwszą iteracją projektu, podczas której ma miejsce wiele czynności wstępnych (patrz sekcja 1.2.5). Testerzy wspólnie z resztą zespołu wykonują następujące czynności w tej iteracji:

- Identyfikacja zakresu projektu (tzn. backlogu produktu).
- Stworzenie początkowej architektury systemu.
- Zaplanowanie, zakup i instalacja niezbędnych narzędzi (np. do zarządzania testami, zarządzania usterkami, automatyzacji testów i do ciągłej integracji).
- Stworzenie początkowej strategii testów dla wszystkich poziomów testów biorącej pod uwagę - między innymi - takie zagadnienia jak: zakres testów, ryzyka techniczne, typy testów (patrz sekcja 3.1.3) oraz cele pokrycia.
- Wykonanie początkowej analizy ryzyk jakościowych (patrz sekcja 3.2.1).
- Zdefiniowanie metryk testowych do mierzenia procesu testowego, postępu testów w projekcie i jakości produktu.
- Określenie definicji ukończenia.
- Utworzenie tablicy zadań (patrz podrozdział 2.2.1).
- Zdefiniowanie, kiedy kontynuować lub zakończyć testy przed dostarczeniem systemu użytkownikom.

Zerowa iteracja nadaje kierunek testom: co testowanie powinno osiągnąć i jak ma to osiągnąć podczas sprintów.

Integracja

W projektach zwinnych, celem jest dostarczanie odbiorcy wartości w sposób ciągły (najlepiej w każdym sprincie). Aby to umożliwić, strategia integracji powinna uwzględniać zarówno projekt jak i testowanie. Żeby zadziałała strategia ciągłego testowania dla dostarczanej funkcjonalności i właściwości, ważne jest określenie wszystkich zależności pomiędzy nimi.

Planowanie testów

Ponieważ testowanie w zespole zwinnym jest w pełni zintegrowane, planowanie testowania powinno rozpoczynać się podczas sesji planowania wydania i być aktualizowane podczas planowania każdego sprintu. Planowanie testów dla wydania oraz dla każdego sprintu powinno obejmować aspekty pokazane w sekcji 1.2.5.

Efektem planowania sprintu jest zbiór zadań do umieszczenia na tablicy. Wykonanie każdego z zadań powinno zabierać jeden lub dwa dni pracy. Dodatkowo wszystkie problemy z testowaniem powinny być śledzone, żeby utrzymywać płynność testowania.

Praktyki testowania zwinnego

Istnieje wiele praktyk, które mogą być użyteczne dla testerów w zespołach scrumowych, np.:

- Praca w parach: Dwóch członków zespołu (np. tester i programista, dwóch testerów lub tester i właściciel produktu) siadają razem przy jednej stacji roboczej żeby testować lub wykonać inne zadanie dla sprintu.
- Przyrostowe projektowanie testów: Przypadki testowe i karty testów są stopniowo budowane z historyjek użytkownika i innych podstaw testów, poczynając od prostych do bardziej skomplikowanych.
- Mapy myśli: Mapy myśli są użytecznym narzędziem przy testowaniu [Crispin08]. Na przykład, testerzy mogą używać mapowania myśli do określenia, które sesje testowe wykonywać, a także do przedstawiania strategii testów i do opisywania danych testowych.

Praktyki te są dodatkiem do innych praktyk opisanych w tym sylabusie oraz opisanych w rozdziale 4 Sylabusa poziomu podstawowego [ISTQB_FL_SYL].

3.2 Ocena ryzyk jakościowych produktu i szacowanie wysiłku testowego

Podstawowym celem testowania we wszystkich projektach (zarówno zwinnych jak i tradycyjnych) jest redukcja ryzyka występowania problemów związanych z jakością produktu, do akceptowalnego poziomu już przed wydaniem. Testerzy w projektach zwinnych mogą wykorzystywać do identyfikowania ryzyk jakościowych, szacowania poziomu ryzyka, oszacowywania nakładu pracy potrzebnego do dostatecznej redukcji tego ryzyka te same rodzaje technik co w projektach tradycyjnych oraz łagodzić ryzyko poprzez projektowanie, implementację i wykonywanie testów. Jednakże, ze względu na krótkie iteracje i dużą ilość zmian w projektach zwinnych niezbędne są pewne adaptacje tych technik.

3.2.1 Szacowanie ryzyk jakościowych produktu w projekcie zwinnym

Jednym z wielu wyzwań w testowaniu jest właściwy dobór, alokacja i priorytetyzacja warunków testowych. Należy do nich określenie stosownego nakładu pracy, wymaganego do pokrycia każdego warunku przypadkami testowymi, a także uporządkowanie uzyskanych przypadków testowych tak, by zoptymalizować efektywność i skuteczność pracy związanej z testowaniem, którą trzeba wykonać. Identyfikacja i analiza ryzyka oraz łagodzenie ryzyka, mogą zostać wykorzystane przez testerów w zespołach zwinnych do oszacowania akceptowalnej ilości przypadków testowych do wykonania. Tym niemniej wiele oddziałujących na siebie ograniczeń i zmiennych może wymagać kompromisów.

Ryzyko jest prawdopodobnym negatywnym lub niepożądanym rezultatem lub zdarzeniem. Poziom ryzyka jest określany przez prawdopodobieństwo wystąpienia ryzyka oraz jego wpływ. Jeżeli podstawowym efektem potencjalnego problemu jest spadek jakości produktu, wówczas potencjalne problemy nazywa się ryzykami jakościowymi lub produktowymi. Jeżeli podstawowym efektem potencjalnego problemu jest zagrożenie powodzenia projektu, wówczas potencjalne problemy nazywa się ryzykami projektowymi lub ryzykami planowania [Black07] [vanVeenendaal12].

W projektach zwinnych, analiza ryzyka jakościowego występuje na dwóch poziomach.

- Podczas planowania wydania: przedstawiciele biznesu znający właściwości do wydania dostarczają wysokopoziomowego opisu ryzyk i cały zespół, razem z testerami, może pomagać w identyfikacji i ocenie ryzyka.
- Podczas planowania iteracji, cały zespół włączając w to testerów identyfikuje i ocenia ryzyka jakościowe produktu.

Przykłady ryzyk jakościowych dla systemu:

- Nieprawidłowe wyliczenia w raportach (ryzyko funkcjonalne odnoszące się do dokładności).
- Wolna reakcja na wprowadzone przez użytkownika dane (ryzyko niefunkcjonalne odnoszące się do efektywności i czasu odpowiedzi).
- Trudności ze zrozumieniem ekranów i pól (niefunkcjonalne ryzyko odnoszące się do użyteczności i zrozumiałości).

Jak wspomniano wcześniej, iteracje rozpoczyna planowanie iteracji, którego rezultatem są oszacowane zadania na tablicy zadań. Zadania te mogą zostać spriorytetyzowane częściowo według ich poziomu ryzyka jakościowego. Zadania posiadające wyższy poziom ryzyka powinny zaczynać się wcześniej i powinny dostać większą pracochłonności testów. Zadania z niższym poziomem ryzyka powinny zaczynać się później i mieć przydzieloną mniejszą pracochłonność testów.

Proces analizy ryzyka jakościowego w projekcie zwinnym może zostać przeprowadzony następującymi krokami:

1. Zbierz razem członków zespołu zwinnego, razem z testerem (testerami).
2. Wypisz wszystkie pozycje z backlogu produktu dla bieżącej iteracji (np. na tablicy zadań).
3. Wypisz wszystkie ryzyka jakościowe związane z każdą pozycją, biorąc pod uwagę wszystkie atrybuty jakościowe.
4. Oceń wszystkie zidentyfikowane ryzyka, co polega na wykonaniu dwóch czynności: kategoryzacji ryzyka oraz ustalenia poziomu ryzyka na podstawie prawdopodobieństwa występowania i wpływu usterek.
5. Ustal dokładność testowania proporcjonalnie do poziomu ryzyka.
6. Wybierz odpowiednie techniki testowania do łagodzenia każdego z ryzyk bazując na ryzyku, jego poziomie oraz odpowiadającemu mu atrybutowi jakości.

Następnie testerzy projektują, implementują oraz wykonują testy, żeby złagodzić ryzyka. Robią to dla całokształtu właściwości, zachowań, atrybutów jakościowych oraz tych wszystkich atrybutów, które wpływają na satysfakcję klientów, użytkowników i interesariuszy.

Podczas trwania projektu, zespół powinien cały czas mieć świadomość, że dodatkowe informacje mogą zmienić zbiór ryzyk lub poziom ryzyk związanych ze znanymi ryzykami jakościowymi. Ponadto zaleca się przeprowadzanie okresowej korekty analizy ryzyk jakościowych produktu, a co za tym idzie odpowiednich zmian w testach. Poprawki zawierają nowe ryzyka, ponowną ocenę poziomów istniejącego ryzyka oraz ocenę skuteczności działań łagodzących ryzyka.

Ryzyko produktowe może być także łagodzone nim rozpoczną się testy. Na przykład, jeżeli podczas identyfikacji ryzyka zostały wykryte problemy z historyjkami użytkownika, zespół projektowy może zastosować gruntowne przeglądy historyjek użytkownika jako działanie łagodzące ryzyko.

3.2.2 Oszacowanie nakładu pracy testowej w oparciu o treść i ryzyko

Podczas planowania wydania zespół zwinny szacuje pracochłonność wymaganą do skompletowania wydania. Szacunki te zawierają również pracochłonność testowania. Często wykorzystywaną techniką przez zespoły zwinne jest poker planistyczny, technika bazująca na konsensusie. Właściciel produktu lub klient czyta historyjkę użytkownika osobom estymującym. Każdy z estymujących ma talię kart z wartościami bliskimi ciągowi Fibonacciego (tj. 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …) lub innemu ciągowi rosnącemu (np. rozmiary koszul od XS do XXL). Wartości na kartach reprezentują ilość punktów historyjek, osobodni lub innych jednostek, w których zespół estymuje. Rekomendowany jest ciąg Fibonacciego, ponieważ liczby w tym ciągu odzwierciedlają wzrost niepewności, która rośnie wraz ze zwiększaniem się rozmiaru historyjki.

Wysokie wyniki szacowania zwykle oznaczają, że historyjka nie została zrozumiana lub, że powinna zostać podzielona na kilka mniejszych.

Estymujący przeprowadzają dyskusję na temat szacowanej właściwości i w ramach potrzeb zadają pytania właścicielowi produktu.

3.3 Techniki w projektach zwinnych

Wiele technik testowych i poziomów testów, które stosuje się w tradycyjnych projektach, może zostać zastosowanych w projektach zwinnych. Niemniej jednak, techniki testowania, terminologia oraz dokumentacja w projektach zwinnych wymagają specyficznego przemyślenia i dostosowania.

3.3.1 Kryteria akceptacji, odpowiednie pokrycie i inne informacje dla testowania

Projekty zwinne przyjmują początkowe wymagania w postaci historyjek użytkownika w uporządkowanym pod względem ważności backlogu na starcie projektu. Początkowo wymagania są krótkie i zgodne z wcześniej zdefiniowanym formatem (patrz Sekcja 1.2.2). Wymagania niefunkcjonalne, takie jak wydajność lub użyteczność, są równie istotne i mogą być specyfikowane jako osobne historyjki użytkownika lub przyłączone do innych funkcjonalnych historyjek użytkownika. Wymagania niefunkcjonalne mogą być zgodne z wcześniej zdefiniowanymi modelami, lub standardami jak [ISO25000], lub ze specyficznymi normami przemysłowymi.

Historyjki użytkownika to główna podstawa testów. Inne możliwe podstawy testów to:

- doświadczenie z poprzednich projektów;
- istniejące funkcje, właściwości i atrybuty jakościowe systemu;
- kod, architektura oraz projekt;
- profile użytkowników (kontekst, konfiguracja systemu i zachowanie użytkownika);
- informacja o defektach z obecnych i poprzednich projektów;
- kategoryzacja usterek w taksonomii defektów;
- dostępne standardy (np. [DO-178B] dla systemów elektroniki lotniczej);
- ryzyka jakościowe (patrz sekcja 3.2.1).

Podczas każdej iteracji programiści tworzą kod, który implementuje, z uwzględnieniem odpowiednich cech jakościowych, funkcje oraz właściwości opisane w historyjkach użytkownika. Kod ten jest weryfikowany i walidowany przez testy akceptacyjne. By kryteria akceptacji były testowalne, muszą poruszać następujące kwestie (o ile są one adekwatne) [Wiegers13]:

- Zachowanie funkcjonalne: Zewnętrznie obserwowalne zachowanie z czynnościami użytkownika, jako wejściami operującymi w pewnych konfiguracjach.

- Cechy jakościowe: Jak system wykonuje określone zachowanie. Cechy jakościowe można również nazywać atrybutami jakościowymi lub wymaganiami niefunkcjonalnymi. Typowe cechy jakościowe to wydajność, niezawodność, użyteczność itp.
- Scenariusze (przypadki użycia): Ciąg akcji pomiędzy zewnętrznym aktorem (często użytkownikiem) i systemem, by osiągnąć określony cel lub zadanie biznesowe.
- Reguły biznesowe: Czynności, które mogą być wykonywane w systemie tylko pod pewnymi warunkami zdefiniowanymi przez zewnętrzne procedury lub ograniczenia, np. procedury stosowane przez firmy ubezpieczeniowe by radzić sobie z roszczeniami.
- Interfejsy zewnętrzne: Opisy połączeń pomiędzy rozwijanym systemem i światem zewnętrznym. Interfejsy zewnętrzne można podzielić na różne typy (interfejs użytkownika, interfejs do innych systemów itp.).
- Ograniczenia: Każde projektowe lub implementacyjne ograniczenie, które zawęża możliwości dewelopera. Urządzenia z oprogramowaniem osadzonym często muszą spełniać wymagania fizyczne takie jak wielkość, waga czy połączenia interfejsów.
- Definicje danych: Klient może opisać format i typ danych, dopuszczalne i domyślne wartości dla elementów danych wchodzących w skład złożonej struktury danych biznesowych (np. kod pocztowy).

Poza historyjkami użytkownika i związanymi z nimi kryteriami akceptacyjnymi, istnieją inne informacje użyteczne dla testera, między innymi:

- Jak system ma pracować i jak będzie używany?
- Interfejsy systemu, które mogą zostać wykorzystane do testowania systemu.
- Czy jest wystarczające wsparcie narzędziowe?
- Czy tester ma wystarczającą wiedzę i umiejętności by wykonać wymagane testy?

Testerzy często odkrywają potrzebę posiadania dodatkowych informacji (np. pokrycie kodu). W trakcie trwania iteracji powinni współpracować z członkami zespołu zwinnego, żeby uzyskać te informacje. Posiadanie odpowiednich informacji gra dużą rolę w określeniu, czy poszczególne zadania mogą zostać uznane za zakończone. Pojęcie definicji ukończenia jest krytyczne dla projektów zwinnych i można je zastosować na wiele różnych sposobów, tak jak jest to opisane w poniższych sekcjach.

Poziomy testów

Każdy poziom testów ma swoja własna definicję ukończenia. Poniższa lista przedstawia przykłady tego kryterium dla różnych poziomów testów.

- Testy modułowe
    - 100% pokrycia decyzji gdzie tylko jest to możliwe, z dokładnym przejrzeniem każdej niewykonalnej ścieżki.
    - Analiza statyczna wykonana dla całego kodu.
    - Brak nierozwiązanych głównych defektów (uporządkowanych zgodnie z priorytetami i wagami).
    - Brak nieakceptowalnego długu technicznego pozostałego w projekcie i kodzie [Jones11].
    - Przejrzany cały kod oraz testy modułowe i ich wyniki.
    - Wszystkie testy modułowe zautomatyzowane.

    - Ważne atrybuty jakościowe znajdują się w wyznaczonych granicach (np. wydajność).
- Testy integracyjne
    - Wszystkie wymagania funkcjonalne przetestowane, włączając w to zarówno testy pozytywne jak i negatywne, z pewną liczbą testów opartych na wielkości, złożoności i ryzykach.
    - Wszystkie interfejsy pomiędzy modułami przetestowane.
    - Pokryte wszystkie ryzyka jakościowe zgodnie z uzgodnionym rozmiarem testów.
    - Rozwiązane wszystkie ważne usterki (uporządkowane zgodnie z priorytetami wg ryzyka i ważności).
    - Zaraportowane wszystkie znalezione defekty.
    - Wszystkie możliwe testy regresyjne są zautomatyzowane i przechowywane we wspólnym repozytorium.
- Testy systemowe
    - Całościowe (end-to-end) testy historyjek użytkownika, właściwości i funkcji.
    - 100% pokrycia ról użytkowników.
    - Pokryte najważniejsze atrybuty jakościowe systemu (np. wydajność, odporność, niezawodność).
    - Testowanie przeprowadzone w środowisku (środowiskach) podobnym do produkcyjnego, w miarę możliwości włączając w to cały sprzęt i oprogramowanie dla wszystkich wspieranych konfiguracji.
    - Wszystkie ryzyka jakościowe pokryte zgodnie z uzgodnionym rozmiarem testów.
    - Zautomatyzowane wszystkie testy regresyjne (tam gdzie tylko jest to możliwe).
    - Wszystkie testy automatyczne przechowywane we wspólnym repozytorium.
    - Wszystkie znalezione defekty zostały zaraportowane i jeżeli to możliwe naprawione.
    - Wszystkie ważne defekty naprawione (uporządkowane zgodnie z priorytetami wg ryzyka i ważności).

Historyjka użytkownika

Poniższe kryteria pokazują przykłady tego, co powinno być spełnione nim historyjka użytkownika dla iteracji uzyska status „ukończona”:

- Historyjki użytkownika wybrane dla iteracji są kompletne, zrozumiane przez zespół oraz mają szczegółowe testowalne kryteria akceptacji.
- Wszystkie części historyki użytkownika zostały spisane i przejrzane, włącznie z testami akceptacyjnymi dla historyjki.
- Zadania dla wybranych historyjek użytkownika zostały zidentyfikowane i oszacowane przez zespół.

Właściwości

Następująca definicja ukończenia może zostać zastosowana dla właściwości, które mogą zawierać kilka historyjek użytkownika lub epików:

- Wszystkie składające się na właściwość historyjki użytkownika, w tym ich kryteria akceptacyjne, są zdefiniowane i zaaprobowane przez klienta.
- Projekt jest kompletny, bez znanego długu technicznego.
- Kod jest kompletny, bez znanego długu technicznego lub niedokończonego refaktoringu.

- Testy modułowe zostały wykonane zgodnie ze zdefiniowanym pokryciem testowym.
- Testy integracyjne i systemowe dla danej właściwości zostały wykonane zgodnie ze zdefiniowanym pokryciem testowym.
- Wszystkie ważne usterki zostały naprawione.
- Dokumentacja danej właściwości jest kompletna, co może oznaczać informację o wydaniu, instrukcję użytkownika i pomoc on-line.

Iteracja

Poniżej podano przykłady definicji ukończenia dla iteracji:

- Wszystkie właściwości z iteracji są gotowe i indywidualnie przetestowane zgodnie z ich kryteriami akceptacji.
- Wszystkie niekrytyczne defekty, które nie mogą zostać naprawione ze względu na ograniczenia iteracji zostały dodane do backlogu produktu i spriorytetyzowane.
- Integracja wszystkich właściwości z iteracji została wykonana i przetestowana.
- Dokumentacja została napisana, przejrzana i zaakceptowana.

W tym punkcie, oprogramowanie jest potencjalnie gotowe do wydania, ponieważ iteracja została zakończona sukcesem, jednak nie wszystkie iteracje kończą się wydaniem.

Wydanie

Poniższa lista dostarcza przykładów kryteriów ukończenia dla wydania, które może rozciągać się na kilka iteracji:

- Pokrycie: Zostały pokryte testami wszystkie konieczne elementy podstawy testów dla całej zawartości wydania. To czy pokrycie jest wystarczające determinowane jest przez to, co jest nowe i zmienione, złożoność i rozmiar tego oraz powiązane z tym ryzyko awarii.
- Jakość: Natężenie defektów (tzn. liczba wykrywanych defektów na dzień lub na transakcję), gęstość defektów (tzn. ilość znalezionych defektów w porównaniu do liczby historyjek użytkownika, pracochłonności lub atrybutów jakościowych) i szacowana liczba pozostałych defektów mieszczą się w akceptowanych granicach. Konsekwencje nierozwiązanych i pozostawionych defektów (np. waga i priorytet) zostały zrozumiane i są akceptowalne. Poziom pozostałego ryzyka związany z każdym zidentyfikowanym ryzykiem jakościowym został zrozumiany i jest akceptowalny.
- Czas: Jeżeli wcześniej określony termin dostawy został osiągnięty, biznesowe czynniki wydania lub nie wydania powinny zostać rozważone.
- Koszt: Oszacowany koszt cyklu życia powinien być wykorzystany do obliczenia zwrotu z inwestycji dla dostarczanego systemu (tzn. obliczony koszt wytworzenia i pielęgnacji powinien być znacząco niższy niż oczekiwany sumaryczny przychód ze sprzedaży produktu). Większa część kosztów w cyklu życia zwykle przypada na koszt pielęgnacji po wdrożeniu, ze względu na liczbę błędów przepuszczonych na produkcję.

3.3.2 Stosowanie wytwarzania sterowanego testami akceptacyjnymi

Ponieważ metoda wytwarzania sterowanego testami akceptacyjnymi jest podejściem “najpierw test”, przypadki testowe są tworzone przed implementacją historyjki użytkownika. Przypadki testowe są tworzone przez zespół zwinny, w skład, którego wchodzi deweloper, tester i przedstawiciel biznesu [Adzic09] i mogą być wykonywane ręcznie lub zautomatyzowane.

Pierwszym krokiem jest warsztat, podczas którego historyjka użytkownika jest analizowana, omawiana i spisywana przez deweloperów, testerów i przedstawicieli biznesu. W tym procesie naprawiane są wszelkie braki, niejednoznaczności oraz błędy.

Następnym krokiem jest utworzenie testów. Może to robić cały zespół, lub sam tester. Tak, czy inaczej, niezależna osoba jak przedstawiciel biznesu zatwierdza przypadki testowe. Testy są przykładami opisującymi specyficzne cechy historyjki użytkownika. Przykłady te pomogą zespołowi poprawnie zaimplementować historyjkę użytkownika. Ponieważ przykłady i testy są jednym i tym samym, używa się tych pojęć naprzemiennie. Praca rozpoczyna się od podstawowych przykładów i otwartych pytań.

Zazwyczaj, pierwsze przypadki testowe są pozytywne. Testują domyślne zachowanie (bez wyjątków lub obsługi błędów) poprzez ciąg akcji wykonywanych, sprawdzających czy wszystko idzie jak oczekiwano. Po tym, jak wykonano testy pozytywne, zespół powinien stworzyć testy negatywne oraz pokryć wymagania niefunkcjonalne (np. wydajność, użyteczność). Testy pisane są tak, by każdy interesariusz był w stanie je zrozumieć. Zawierają zdania budowane w języku naturalnym, jednocześnie podając konieczne warunki wstępne (jeśli takowe występują), wejścia oraz odpowiadające im wyjścia.

Przykłady muszą pokrywać wszystkie cechy historyjki użytkownika, ale nie powinny nic dodawać do historyjki. Oznacza to, że nie powinien istnieć żaden przykład, który opisuje taki aspekt historyjki użytkownika, który nie jest w niej opisany. Co więcej żadne dwa przykłady nie powinny opisywać tej samej cechy historyjki użytkownika.

3.3.3 Czarnoskrzynkowe techniki projektowania testów funkcjonalnych i niefunkcjonalnych

W testowaniu zwinnym wiele testów jest tworzonych równolegle z tworzeniem oprogramowania. Tak samo jak programiści tworzą oprogramowanie w oparciu o historyjki użytkownika i kryteria akceptacji, testerzy tworzą testy również w oparciu o historyjki użytkownika i kryteria akceptacji. Niektóre testy, takie jak testy eksploracyjne oraz inne testy oparte na doświadczeniu są tworzone później podczas fazy wykonania testów (patrz podrozdział 3.3.4). Testerzy mogą wykorzystywać do projektowania testów tradycyjne techniki czarnoskrzynkowe, jak klasy równoważności, analiza wartości brzegowych, tablice decyzyjne oraz testy przejść między stanami. Na przykład analiza wartości brzegowych może zostać użyta do wyboru wartości testowych, gdy klient ma ograniczoną liczbę rzeczy, które może wybrać podczas zakupu.

W wielu sytuacjach wymagania niefunkcjonalne mogą także być wyrażane, jako historyjki użytkownika. Czarnoskrzynkowe techniki projektowania testów (np. analiza wartości brzegowych) mogą zostać wykorzystane do tworzenia testów niefunkcjonalnych atrybutów jakościowych. Historyjka użytkownika może zawierać wymagania na wydajność lub niezawodność. Na przykład, dane wykonanie nie może przekroczyć określonego czasu lub liczba operacji, które mogą się nie powieść ma być mniejsza niż podana wartość.
Więcej informacji na temat wykorzystania technik czarnoskrzynkowych znajduje się w sylabusie Poziomu Podstawowego [ISTQB_FL_SYL] oraz sylabusie Poziomu Zaawansowanego dla Analityka Testowego [ISTQB_ALTA_SYL].

3.3.4 Testowanie eksploracyjne a testowanie zwinne

Testowanie eksploracyjne pełni bardzo ważną rolę w projektach zwinnych ze względu na ograniczony czas na analizę testów oraz znikomą ilość szczegółów w historyjkach użytkownika.

Najlepsze rezultaty osiąga się, gdy testowanie eksploracyjne uzupełnia się innymi technikami opartymi na doświadczeniu, jako części reaktywnej strategii testów, którą łączy się z innymi podejściami do testów, takimi jak analityczne testowanie oparte na ryzyku, analityczne testowanie oparte na wymaganiach, testowanie oparte na modelu czy testowanie przeciwregresywne. Strategie testowania oraz ich łączenie opisane są w sylabusie Poziomu Podstawowego [ISTQB_FL_SYL].
W testowaniu eksploracyjnym projektowanie i wykonywanie testów wykonuje się równocześnie, w oparciu o wcześniej przygotowaną kartę testów. Karta testów dostarcza warunków testowych, które powinny zostać pokryte podczas ograniczonej czasowo sesji testowej. Podczas testów eksploracyjnych, wyniki ostatnich testów prowadzą do następnych testów. Te same techniki biało- i czarnoskrzynkowe, których używa się przy wykonywaniu testów zaprojektowanych, mogą być używane również w testach eksploracyjnych.
Karta testów może zawierać następujące informacje:

- Aktor: Zamierzony użytkownik systemu.
- Cel: Temat karty, w tym szczególny cel, jaki aktor chce osiągnąć, tzn. warunki testowe.
- Konfiguracja: Co powinno być przygotowane, by rozpocząć wykonywanie testów.
- Priorytet: Względna istotność tej karty, oparta na priorytecie powiązanej historyjki użytkownika lub poziomie ryzyka.
- Odnośniki: Specyfikacja (np. historyjka użytkownika), ryzyko i inne źródła informacji.
- Dane: Jakiekolwiek dane potrzebne do wykonania karty.
- Czynności: Lista pomysłów na to, co aktor może chcieć wykonywać w systemie (np. “Zalogować się do systemu jako superużytkownik”), a także co mogłoby być intersujące do przetestowania (zarówno w testach pozytywnych jak i negatywnych).
- Uwagi wyroczni testowej: Jak oceniać produkt, by określić poprawne wyniki (np. uchwycić co pojawia się na ekranie i porównać to z tym, co napisano w instrukcji użytkownika).
- Wariacje: Alternatywne akcje i oceny uzupełniające pomysły opisane w czynnościach.

Do zarządzania testami eksploracyjnymi wykorzystuje się metodę zwaną zarządzaniem testami w sesjach. Sesja to nieprzerywalny okres testowania trwający od 60 do 120 minut. Sesje testowe można podzielić następująco:

- sesja sondażowa (by nauczyć się, jak to działa);
- sesja analityczna (ocena funkcjonalności lub właściwości);
- głębokie pokrycie (skrajne przypadki, scenariusze, interakcje).

Jakość testów zależy od zdolności testera do zadawania właściwych pytań o to, co testować. Przykłady:

- Co jest najważniejsze do odkrycia w systemie?
- W jaki sposób system może ulec awarii?
- Co się stanie, gdy …?
- Co powinno się stać, kiedy …?

- Czy spełnione są potrzeby, wymagania i oczekiwania użytkownika?
- Czy można system zainstalować (i usunąć, gdy jest to konieczne) na wszystkich wspieranych ścieżkach aktualizacyjnych?

Podczas wykonywania testów tester wykorzystuje swoją kreatywność, intuicję, percepcję oraz umiejętności do wyszukiwania potencjalnych problemów w produkcie. Tester powinien także posiadać szeroką wiedzę na temat systemu, dobrze rozumieć testowane oprogramowanie, dziedzinę biznesową, sposób w jaki system jest używany oraz jak stwierdzić, że dany system działa niepoprawnie.

Podczas testowania można zastosować zbiór heurystyk. Heurystyka to praktyczna zasada, dająca testerowi pewne wskazówki jak wykonywać testy i oceniać wyniki [Jeffries00]. Przykłady heurystyk to:

- granice;
- CRUD (Create, Read, Update, Delete - utwórz, odczytaj, aktualizuj i usuń);
- różne konfiguracje;
- przerwania (np. wylogowanie, zamykanie, restart);

Ważne jest żeby tester dokumentował proces testów tak dokładnie jak jest to możliwe. Bez tego powrót i odkrycie sposobu wykrycia problemu w systemie może być znacząco utrudnione. Poniższa lista zawiera przykłady informacji, które warto zapisać:

- Pokrycie testowe: Jakich danych wejściowych użyto, ile pokrywają i co pozostało do testowania.
- Notatki z oceny: Obserwacje z testów, czy testowany system i testowana właściwość jest stabilna, czy znaleziono jakieś defekty, jaki jest planowany następny krok zgodnie z bieżącą obserwacją i lista pomysłów.
- Lista ryzyk/strategii: Które z najważniejszych ryzyk zostały pokryte, a które pozostały; czy kontynuuje się początkową strategię, czy wymaga ona jakiś zmian.
- Problemy, zapytania i anomalie: Wszystkie nieoczekiwane zachowania, pojawiające się pytania dotyczące efektywności podejścia, jakiekolwiek zastrzeżenia dotyczące pomysłów/podejścia do testów, środowiska testowego, danych testowych, niezrozumienia funkcji, skryptu testowego lub testowanego systemu.
- Rzeczywiste zachowanie: Zapisy rzeczywistego zachowania systemu, które należy zachować (np. wideo, zrzuty ekranów, pliki danych wynikowych).

Zebrane informacje powinny być zapisywane lub podsumowywane w jakimś narzędziu do zarządzania statusem testów (np. narzędziu do zarządzania testami, narzędziu do zarządzania zadaniami lub tablicy zadań).

3.4 Narzędzia w projektach zwinnych

Narzędzia przedstawione w sylabusie poziomu podstawowego [ISTQB-FL_SYL] są wciąż odpowiednie i używane przez testerów w zespołach zwinnych. Nie wszystkie są wykorzystywane w ten sam sposób, a niektóre z nich pasują bardziej do projektów zwinnych niż w tradycyjnych. Na przykład, narzędzia do zarządzania testami, zarządzania wymaganiami czy zarządzania incydentami (narzędzia do śledzenia defektów) mogą być używane przez zespoły zwinne. Część tych zespołów optuje za narzędziami zawierającymi wszystko (np. narzędziami do zarządzania cyklem życia aplikacji lub zarządzania zadaniami), które posiadają cechy odpowiednie do wytwarzania zwinnego, takie jak tablice zadań, wykresy spalania, czy historyjki użytkownika. Dla testerów w zespołach zwinnych ważne są narzędzia do zarządzania konfiguracją ze względu na dużą liczbę automatyzowanych testów na wszystkich poziomach, a także potrzebę przechowywania i zarządzania powiązanymi z nimi artefaktami.

Jako uzupełnienie do narzędzi opisanych w sylabusie poziomu podstawowego [ISTQB-FL_SYL] testerzy w projektach zwinnych używają też narzędzi opisanych w poniższych podrozdziałach. Narzędzia te, używane przez cały zespół, zapewniają współpracę w zespole i dzielenie się informacją, co jest kluczowe dla podejścia zwinnego.

3.4.1 Narzędzia do zarządzania zadaniami i śledzenia

W pewnych przypadkach, zespoły zwinne używają fizycznych tablic (np. białych lub korkowych tablic) do zarządzania i śledzenia historyjek użytkownika, testów i innych zadań podczas każdego sprintu. Inne zespoły wykorzystują oprogramowanie do zarządzania cyklem życia aplikacji i zarządzania zadaniami, w tym elektroniczne tablice zadań. Narzędzia te mogą służyć do następujących celów:

- Zapisywanie historyjek i związanych z nimi zadań wytwórczych i testowych, by mieć pewność, że nic nie zginęło podczas sprintu.
- Zbieranie wyników szacowania zadań przez członków zespołu i automatyczne obliczanie nakładu pracy, potrzebnego do zaimplementowania historyjki, wspomagając efektywnie sesje planowania iteracji.
- Powiązanie zadań wytwórczych i testowych dla tej samej historyjki, by dostarczyć pełny obraz nakładu pracy zespołu niezbędnego do jej implementacji.
- Agregowanie raportów programistów i testerów, co do statusu zadań w momencie ich zakończenia, jednocześnie dostarczając aktualnego statusu każdej historyjki, iteracji i całego wydania.
- Dostarczają wizualnego przedstawienia (z użyciem metryk, wykresów, tablic rozdzielczych) aktualnego stanu każdej historyjki użytkownika, iteracji oraz wydania, umożliwiając wszystkim interesariuszom, w tym ludziom z geograficznie rozproszonych zespołów szybki dostęp do statusu.
- Integrację z narzędziami do zarządzania konfiguracją, co może pozwolić na automatyczne zapisywanie wgranego na serwer kodu (ang. „commit") i wersji w stosunku do zadań oraz - w pewnych sytuacjach - automatycznej aktualizacji statusu dla zadań.

3.4.2 Narzędzia do komunikacji i wymiany informacji

Jako uzupełnienie do komunikacji e-mailowej, przy pomocy dokumentów i komunikacji werbalnej, zespoły zwinne często wykorzystują trzy dodatkowe typy narzędzi by wspomóc komunikację i dzielenie się informacjami: wiki, komunikatory i współdzielenie pulpitu.

Wiki umożliwia zespołom budowę i współdzielenie bazy wiedzy o różnych aspektach projektu, zawierającej m.in.:

- Diagramy właściwości produktu, dyskusje o właściwościach, diagramy prototypów, zdjęcia tablic z zapisami dyskusji i inne informacje.
- Informacje o narzędziach lub technikach wytwarzania i testowania, które zostały uznane za użyteczne przez innych członków zespołu.
- Metryki, wykresy i tablice rozdzielcze ze statusem produktu; jest to szczególnie użyteczne, gdy wiki jest zintegrowana z innymi narzędziami, takimi jak serwer budowania wersji, czy też system zarządzania zadaniami. Może on wówczas automatycznie aktualizować status produktu.
- Rozmowy pomiędzy członkami zespołu; podobnie jak komunikatory oraz e-mail, ale wiki pozwala na dzielenie się nimi ze wszystkimi w zespole.

Komunikatory, narzędzia do konferencji audio i wideo dostarczają następujących korzyści:

- Umożliwiają bezpośrednią komunikację w czasie rzeczywistym pomiędzy członkami zespołu, co jest ważne zwłaszcza w zespołach rozproszonych.
- Angażują zespoły rozproszone podczas codziennych spotkań.
- Zmniejszają rachunki telefoniczne dzięki wykorzystaniu technologii VOIP, niwelując problem ograniczania kosztów, co mogłoby wpłynąć na osłabienie komunikacji członków rozproszonych zespołów.

Narzędzia do współdzielenia pulpitu i przechwytywania dostarczają następujących korzyści:

- W zespołach rozproszonych umożliwiają demonstrowanie produktu, przeglądy kodu, a nawet pracę w parach.
- Zapisywanie nagrań z demonstracji produktu na końcu każdej iteracji i przechowywanie ich na wiki zespołu.

Narzędzia te powinny być używane, jako uzupełnienie i rozszerzenie, ale nie zastąpienie, komunikacji twarzą w twarz w zespołach zwinnych.

3.4.3 Narzędzia do budowy i dystrybucji

Jak przedyskutowano wcześniej w tym sylabusie, codzienne budowanie i wdrażanie oprogramowania jest kluczową praktyką w zespołach zwinnych. Wymaga to używania narzędzi do ciągłej integracji i narzędzi do budowania dystrybucji. Możliwe sposoby wykorzystania, korzyści oraz ryzyka związane z tymi narzędziami zostały opisane w sekcji 1.2.4.

3.4.4 Narzędzia do zarządzania konfiguracją

W zespołach zwinnych, narzędzia do zarządzania konfiguracją mogą być używane nie tylko do przechowywania kodu źródłowego i testów automatycznych, lecz również testów manualnych, a także innych produktów pracy testerskiej, które nierzadko są przechowywane w tym samym repozytorium, co kod źródłowy produktu. Umożliwia to śledzenie, które wersje oprogramowania zostały pokryte którymi wersjami testów i pozwala na szybkie zmiany bez utraty informacji o historii. W skład podstawowych typów systemów kontroli wersji wchodzą zarówno systemy scentralizowane jak i rozproszone. Wielkość zespołu, jego struktura, lokalizacja i wymagania dotyczące integracji z innymi narzędziami określają, jaka wersja narzędzia do kontroli wersji najlepiej odpowiada danemu zespołowi zwinnemu.

3.4.5 Narzędzia do projektowania, implementacji i wykonywania testów

Pewne narzędzia są użyteczne dla testera zwinnego w określonych punktach procesu testowania oprogramowania. Jakkolwiek większość z tych narzędzi nie jest nowych ani specyficznych dla procesu zwinnego, dostarczają one istotnych możliwości biorąc pod uwagę gwałtowne i częste zmiany w projektach zwinnych.

- Narzędzia do projektowania testów: Użycie narzędzi takich jak mapy myśli staje się coraz popularniejsze do szybkiego projektowania i definiowania testów dla nowych właściwości.
- Narzędzia do zarządzania przypadkami testowymi: Narzędzi do zarządzania przypadkami testowymi w projektach zwinnych mogą być częścią narzędzia – będącego własnością całego zespołu - do zarządzania cyklem życia lub zarządzania zadaniami.
- Narzędzia przygotowujące lub generujące dane testowe: Narzędzia, które generują dane wypełniające bazę danych aplikacji przy pomocy różnych skryptów przynoszą duże korzyści, kiedy duża liczba danych i ich kombinacji jest niezbędna do testowania aplikacji. Narzędzia te mogą także pomagać przy przedefiniowywaniu struktury bazy danych, gdy następują zmiany podczas projektu zwinnego i refaktoryzacji skryptów generujących dane. Umożliwia to szybką aktualizację danych testowych, gdy pojawiają się zmiany. Niektóre narzędzia do przygotowywania danych testowych używają danych produkcyjnych jako materiału źródłowego, a także wykorzystują skrypty do usuwania lub anonimizacji danych wrażliwych. Inne narzędzia przygotowujące dane testowe są pomocne przy walidacji dużych zbiorów danych wejściowych lub wyjściowych.
- Narzędzia do ładowania danych testowych: Po wygenerowaniu danych do testowania, powinny one zostać załadowane do aplikacji. Ręczne wprowadzanie danych zabiera dużo czasu i umożliwia powstanie błędów, na szczęście istnieją odpowiednie narzędzia do ładowania danych. Wiele z narzędzi do automatycznego tworzenia danych zawiera zintegrowany komponent do ładowania danych. W pozostałych przypadkach możliwe jest masowe ładowanie przy użyciu systemów zarządzania bazami danych.
- Narzędzia do automatycznego wykonywania testów: Istnieją narzędzia do wykonywania testów, które z zasady bardziej pasują do zespołów zwinnych. Narzędzia te, zarówno komercyjne jak i o otwartym kodzie, wspomagają podejście „najpierw test”, to znaczy wytwarzanie sterowane zachowaniem (BDD), wytwarzanie sterowane testami (TDD) czy wytwarzanie sterowane testami akceptacyjnymi (ATDD). Narzędzia te umożliwiają testerom i biznesowi wyrażanie oczekiwanego zachowania systemu w tabelach lub w języku naturalnym z użyciem słów kluczowych.

- Narzędzia do testów eksploracyjnych: Narzędzia, które rejestrują i logują czynności wykonywane na aplikacji podczas sesji testów eksploracyjnych są użyteczne zarówno dla testera jak i dewelopera, ponieważ zapisują one wykonane czynności. Jest to pomocne, gdy zostaje znaleziona usterka, ponieważ czynności wykonane przed awarią zostały zarejestrowane i mogą być wykorzystane przy raportowaniu usterki deweloperom. Logowanie kroków wykonanych w sesji testów eksploracyjnych może się okazać bardzo korzystne, jeżeli test będzie docelowo włączony to zestawu automatycznych testów regresji.

3.4.6 Narzędzia do przetwarzania w chmurze i do wirtualizacji

Wirtualizacja umożliwia pojedynczemu zasobowi fizycznemu (serwerowi) działanie, jako wiele oddzielnych mniejszych zasobów. Gdy używa się maszyn wirtualnych bądź w chmurze, zespół ma większą liczbę serwerów dostępnych dla wytwarzania i testowania. Umożliwia to uniknięcie opóźnienia związanego z oczekiwaniem na fizyczne serwery. Postawienie nowego serwera lub przywrócenie stanu serwera jest bardziej efektywne, gdy możliwe jest wykonanie obrazów stanu, co potrafi większość wirtualnych narzędzi. Niektóre narzędzia do zarządzania testami wykorzystują technologię wirtualizacji do zrzucenia obrazu serwera w momencie, w którym wykryto defekt, umożliwiając testerom podzielenie się tym zrzutem z deweloperami analizującymi usterkę.

4 Bibliografia

4.1 Standardy

- [DO-178B] RTCA/FAA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992.

- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE), 2005.

4.2 Dokumenty ISTQB

- [ISTQB_ALTA_SYL] ISTQB Advanced Level Test Analyst Syllabus, Version 2012
- [ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, Version 2012
- [ISTQB_FA_OVIEW] ISTQB Foundation Level Agile Tester Overview, Version 1.0
- [ISTQB_FL_SYL] ISTQB Foundation Level Syllabus, Version 20

4.3 Książki

[Aalst13] Leo van der Aalst and Cecile Davis, “TMap NEXT® in Scrum,” ICT-Books.com, 2013.
[Adzic09] Gojko Adzic, “Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing,” Neuri Limited, 2009.
[Anderson13] David Anderson, “Kanban: Successful Evolutionary Change for Your Technology Business,” Blue Hole Press, 2010.
[Beck02] Kent Beck, “Test-driven Development: By Example,” Addison-Wesley Professional, 2002.
[Beck04] Kent Beck and Cynthia Andres, “Extreme Programming Explained: Embrace Change, 2e” Addison-Wesley Professional, 2004.
[Black07] Rex Black, “Pragmatic Software Testing,” John Wiley and Sons, 2007.
[Black09] Rex Black, “Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 3e,” Wiley, 2009.
[Chelimsky10] David Chelimsky et al, “The RSpec Book: Behavior Driven Development with Rspec, Cucumber, and Friends,” Pragmatic Bookshelf, 2010.
[Cohn04] Mike Cohn, “User Stories Applied: For Agile Software Development,” Addison-Wesley Professional, 2004.
[Crispin08] Lisa Crispin and Janet Gregory, “Agile Testing: A Practical Guide for Testers and Agile Teams,” Addison-Wesley Professional, 2008.
[Goucher09] Adam Goucher and Tim Reilly, editors, “Beautiful Testing: Leading Professionals Reveal How They Improve Software,” O'Reilly Media, 2009.

[Jeffries00] Ron Jeffries, Ann Anderson, and Chet Hendrickson, “Extreme Programming Installed,” Addison-Wesley Professional, 2000.
[Jones11] Capers Jones and Olivier Bonsignour, “The Economics of Software Quality,” Addison-Wesley Professional, 2011.
[Linz14] Tilo Linz, “Testing in Scrum: A Guide for Software Quality Assurance in the Agile World,” Rocky Nook, 2014.
[Schwaber01] Ken Schwaber and Mike Beedle, “Agile Software Development with Scrum,” Prentice Hall, 2001.
[vanVeenendaal12] Erik van Veenendaal, “The PRISMA approach”, Uitgeverij Tutein Nolthenius, 2012.
[Wiegers13] Karl Wiegers and Joy Beatty, “Software Requirements, 3e,” Microsoft Press, 2013.

4.4 Terminologia zwinna

Słowa kluczowe, które można znaleźć w słowniku ISTQB zostały wymienione na początku każdego rozdziału. Często spotykane pojęcia zwinne zostały zaczerpnięte z następujących, ogólnie akceptowanych źródeł internetowych, które podają definicje:

http://guide.agilealliance.org/
http://searchsoftwarequality.techtarget.com
http://whatis.techtargetcom/glossary
http://www.scrumalliance.org/

Zachęcamy czytelników do odwołania się do w/w stron WWW, jeżeli nie znają terminologii związanej z metodykami zwinnymi używanej w tym dokumencie. Podane linki były aktywne w momencie tworzenia tego dokumentu.

Polskie słownictwo związane z metodykami zwinnymi zostało zaczerpnięte z Przewodnika po Scrumie w tłumaczeniu Tomasza Włodarka i innych [Scrum Guide].

4.5 Inne pozycje

Poniższe odnośniki wskazują na informacje dostępne w Internecie i w innych miejscach. Miejsca te zostały sprawdzone w momencie publikacji tego sylabusa. Niemniej jednak ISTQB nie jest odpowiedzialne za utrzymywanie ich dostępności.

Rozdział 3:

- Czerwonka, Jacek: www.pairwise.org
- Bug Taxonomy: www.testingeducation.org/a/bsct2.pdf

5. Indeks