Certyfikowany Tester
Sylabus Poziomu Podstawowego
Agile
Tester Zwinny

v 1.0 (2014)
PL w. 1.0.0.6 (2024)

International Software Testing Qualifications Board®

Prawa autorskie

Grupa robocza Foundation Level Extension Agile Tester: Rex Black (Przewodniczący), Bertrand Cornanguer (Wiceprzewodniczący),Gerry Coleman (Lider Celów Nauczania), Debra Fredenberg (Lider Pytań Egzaminacyjnych), Alon Linetzki (Lider ds. Marketingu oraz Korzyści Biznesowych), Tauhida Parveen (Edytor) i Leo van der Aalst (Lider Wdrożenia).

Autorzy: Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh i 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 i 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 Kazun. Końcowa edycja: Lucjan Stapp.

Redakcja tekstu i korekta tłumaczenia (2022): Monika Petri-Starego i Lucjan Stapp.

Redakcja tekstu i korekta tłumaczenia (2024): Monika Petri-Starego.
Przegląd (2024): Adam Roman i Lucjan Stapp.

Historia zmian

Wersja	Data	Uwagi
sylabus PL w. 1.0.0.6 (2024)	13.02.2024 r.	Przegląd całości dokumentu, korekta tłumaczenia.
sylabus PL w. 1.0.0.5. (2023)	16.06.2023 r.	Zmiana tłumaczenia terminu (podejścia) z „najpierw testuj” na „najpierw test”, zgodnie ze Słownikiem terminów testowych ISTQB® (zmianę wprowadzono w: 1.2.1., 3.3.2., 3.4.5.).
sylabus PL w. 1.0.0.4 (2023)	05.06.2023 r.	Zmiana zasad numeracji sylabusa (czwarta cyfra to numer wersji polskiej). Zmiana tłumaczenia terminu z „tablica rozdzielcza” na „tablica wskaźników” (w: 1.2.4., 2.1.2., 2.2.1., 3.4.1., 3.4.2.).
sylabus v 1.0 (2014) PL w. 1.0.3 (2022)	15.10.2022 r.	Zmiana zapisów dotyczących praw autorskich. Korekta tłumaczenia. Aktualizacja nazw poziomów certyfikacji. Aktualizacja terminologii.
sylabus 2014 PL w. 1.0.2 (2021)	24.02.2021 r.	Przegląd edytorski.
sylabus 2014 PL w. 1.0.1 (2014)	30.11.2014 r.	Usunięcie literówek i niezgodności wskazanych przez użytkowników.
sylabus 2014 PL w. 1.0 (2014)	24.10.2014 r.	Zatwierdzenie przez Zarząd SJSI.
sylabus 2014	31.05.2014 r.	Zatwierdzenie przez GA.

Spis treści

Podziękowania

Dokument ten został napisany przez zespół International Software Testing Qualifications Board Foundation Level Working Group.

Zespół ds. Rozszerzenia Zwinnego dziękuje zespołowi recenzentów oraz wszystkim Radom Krajowym za ich sugestie i uwagi.
W momencie ukończenia sylabusa Rozszerzenie Poziomu Podstawowego – Tester Zwinny[1] Grupa Robocza ds. Rozszerzenia Zwinnego składała się z następujących osób: Rex Black (Przewodniczący), Bertrand Cornanguer (Wiceprzewodniczący), Gerry Coleman (Lider Celów Nauczania), Debra Fredenberg (Lider Pytań Egzaminacyjnych), Alon Linetzki (Lider ds. Marketingu oraz Korzyści Biznesowych), Tauhida Parveen (Edytor) i Leo van der Aalst (Lider ds. Rozwoju).

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

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

Zespół dziękuje również następującym osobom z Rad Krajowych oraz społeczności ekspertów Agile, którzy bral i udział w przeglądaniu, komentowaniu oraz głosowaniu nad sylabusem Rozszerzenie Poziomu Podstawowego – Tester Zwinny[2] (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 publikacji przez Zgromadzenie Ogólne ISTQB® w dniu 31 maja 2014 r.

[1] Aktualna nazwa tego poziomu: Certyfikowany Tester – Poziom Podstawowy Agile – Tester Zwinny
[2] Patrz przypis nr 1.

0 Wstęp

0.1 Cel

Niniejszy sylabus stanowi podstawę dla międzynarodowej kwalifikacji w testowaniu oprogramowania na Poziomie Podstawowym Agile – Tester Zwinny. ISTQB® udostępnia ten sylabus w następujący sposób:

- Radom Krajowym w celu przetłumaczenia na ich język lokalny i akredytacji dostawców szkoleń. Rady Krajowe mogą dostosować ten sylabus do swoich potrzeb językowych oraz zmodyfikować referencje tak, aby zawierały lokalne publikacje.
- Organom certyfikacyjnym w celu opracowania pytań egzaminacyjnych w ich lokalnym języku, dostosowanych do celów nauczania dla każdego sylabusa.
- Dostawcom szkoleń w celu opracowania materiałów szkoleniowych i określenia odpowiednich metod nauczania.
- Kandydatom do certyfikacji w celu przygotowania się do egzaminu (w ramach szkoleń lub niezależnie od nich).
- Międzynarodowej społeczności twórców oprogramowania i systemów w celu rozwoju zawodu tester oprogramowania i systemów oraz jako podstawę dla książek i artykułów.

ISTQB® może zezwolić również innym podmiotom na wykorzystywanie sylabusa do innych celów pod warunkiem uzyskania uprzedniej pisemnej zgody.

0.2 Opis

Dokument Poziom Podstawowy Agile - Tester Zwinny Przegląd A [ISTQB_FA_OVIEW] zawiera następujące informacje:

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

0.3 Cele nauczania podlegające egzaminowaniu

Cele nauczania wynikają z korzyści biznesowych i są wykorzystywane do opracowania egzaminu Certyfikowany Tester – Poziom Podstawowy Agile – Tester Zwinny. Wszystkie części tego sylabusa są sprawdzane na egzaminie co najmniej na poziomie K1. Oznacza to, że kandydat powinien rozpoznać, zapamiętać pojęcie lub koncepcję. Konkretne cele nauczania na poziomach K1, K2 i K3 są przedstawione na początku odpowiedniego rozdziału.

1 Zwinne wytwarzanie oprogramowaniaBłąd! Nie zdefiniowano zakładki. - 150 minut

Słowa kluczowe

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

Cele nauczania w zakresie 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 programistami i z przedstawicielami biznesu.

FA-1.2.3 (K2) Kandydat rozumie, w jaki sposób retrospektywy mogą być wykorzystywane jako mechanizm doskonalenia procesów w projektach zwinnych.

FA-1.2.4 (K2) Kandydat rozumie zastosowanie i cel ciągłej integracji.

FA-1.2.5 (K1) Kandydat zna różnice pomiędzy planowaniem iteracji i wydań oraz wie, w jaki sposób tester wnosi wartość dodaną do każdej z tych czynności.

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 leżące u podstaw projektów zwinnych oraz to, że testerzy są integralną częścią podejścia „cały zespół wraz z programistami i przedstawicielami jednostek biznesowych. Członkowie zespołów zwinnych komunikują się ze sobą wcześnie i często, co pomaga we wczesnym usuwaniu defektów i tworzeniu produktu 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 deklaracje wartości:

- 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 przekonuje, że chociaż koncepcje wymienione po prawej stronie mają wartość, to jednak uznaje, iż te po lewej mają większą wartość.

Ludzie i współpraca

Wytwarzanie zwinne jest mocno 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 zbyt szczegółowa dokumentacja, ponadto zapewnia ono szansę na szybkie dostarczenie zespołowi wytwórczemu informacji zwrotnej. Co więcej, ponieważ działające oprogramowanie, nawet z ograniczoną funkcjonalnością, jest dostępne na znacznie wcześniejszym etapie cyklu życia wytwarzania oprogramowania, wytwarzanie zwinne może mieć wcześniejszy termin wdrożenia (ang. *time to market*). Wytwarzanie zwinne jest zatem szczególnie użyteczne w szybko zmieniających się środowiskach biznesowych, 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 systemowych. Dzięki bezpośredniej współpracy z klientem, wzrastają szanse na zrozumienie, czego klient tak naprawdę potrzebuje. Chociaż posiadanie umów z klientami może być ważne, regularna i ścisła współpraca z nimi prawdopodobnie doprowadzi do zakończenia projektu sukcesem.

Reagowanie na zmiany

Zmiany są nieuniknione w większości projektów informatycznych. Środowisko, w którym działa firma, ustawodawstwo, działalność konkurencji, postęp technologiczny oraz inne czynniki mogą mieć duży wpływ na projekt i jego cele. Czynniki te muszą być uwzględnione w procesie wytwarzania. W związku z tym, zachowanie elastyczności w sposobie pracy, tak aby radzić sobie ze zmianami, jest istotniejsze niż sztywne trzymanie się planu.

Zasady

Podstawowe wartości Manifestu Zwinnego Wytwarzania Oprogramowania zostały ujęte w 12 zasadach:

- Dla zespołu najwyższy priorytet ma zadowolenie klienta wynikające z wczesnego i ciągłego dostarczania wartościowego oprogramowania.
- Zespół akceptuje, że wymagania mogą się zmienić nawet na późnym etapie projektu. Zwinne procesy wykorzystują zmiany dla uzyskania przewagi konkurencyjnej przez klienta.
- Działające oprogramowanie jest dostarczane często, najlepiej w odstępach od kilku tygodni do kilku miesięcy, im częściej, tym lepiej.
- Przedstawiciele jednostek biznesowych i programiści muszą współpracować codziennie przez cały czas trwania projektu.
- Projekty tworzone są wokół zmotywowanych osób. Należy zapewnić im odpowiednie środowisko i wsparcie. Należy im zaufać, że dobrze wykonają swoją pracę.
- Najwydajniejszym i najskuteczniejszym sposobem przekazywania informacji do i w ramach zespołu wytwórczego jest bezpośrednia rozmowa (twarzą w twarz).
- Podstawową i najważniejszą miarą postępu jest działające oprogramowanie.
- Procesy zwinne umożliwiają zrównoważony rozwój. Sponsorzy, programiści oraz użytkownicy powinni być w stanie utrzymywać równe tempo pracy.
- Ciągłe skupienie na technicznej doskonałości i dobrym projekcie oprogramowania zwiększa zwinność.
- Prostota – sztuka maksymalizacji ilości niewykonanej pracy – jest zasadnicza.
- Najlepsze rozwiązania architektoniczne, wymagania i projekty powstają w samoorganizujących się zespołach.
- Zespół zastanawia się w regularnych odstępach czasu nad tym, jak poprawić swoją efektywność, a następnie dostosowuje lub zmienia swoje działanie.

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

1.1.2 Podejście „cały zespół”

Podejście „cały zespół” oznacza zaangażowaniewszystkich osób posiadających wiedzę i umiejętności niezbędne do zapewnienia sukcesu projektu. Zespół obejmuje przedstawicieli klienta i innych interesariuszy biznesowych, którzy określają cechy funkcjonalne 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 (ang. *stand-up meetings*, patrz sekcja 2.2.1) z udziałem wszystkich członków 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 korzyścią dla projektu;
- czyni wszystkich odpowiedzialnymi za jakość.

W projekcie zwinnym za jakość odpowiada cały zespół. Istotą podejścia „cały zespół" jest współpraca testerów, programistów oraz przedstawicieli jednostek biznesowych na każdym etapie procesu wytwarzania oprogramowania. Testerzy ściśle współpracują zarówno z programistami, jak i z przedstawicielami jednostek biznesowych, by zapewnić osiągnięcie pożądanego poziomu jakości. To współdziałanie obejmuje: pomaganie i współpracę z przedstawicielami jednostek biznesowych, by wesprzeć ich w tworzeniu odpowiednich testów akceptacyjnych; współpracę z programistami w celu uzgodnienia strategii testów oraz podejmowanie decyzji dotyczących podejścia do automatyzacji testów. Testerzy mogą w ten sposób przekazywać i rozszerzać wiedzę z zakresu testowania na innych członków zespołu oraz wpływać na wytwarzanie produktu.

Cały zespół jest zaangażowany w konsultacje lub spotkania, na których są przedstawiane, analizowane lub oceniane cechy funkcjonalne produktu. Koncepcja angażowania testerów, programistów i przedstawicieli jednostek biznesowych we wszystkie dyskusje związane z cechami funkcjonalnymi nazywana jest siłą trzech (ang. *the power of three*) [Crispin08].

1.1.3 Wczesna i częsta informacja zwrotna

Projekty zwinne mają krótkie iteracje umożliwiające zespołowi projektowemu otrzymywanie wczesnych i częstych informacji zwrotnych dotyczących jakości produktu przez cały cykl wytwarzania. Jednym ze sposobów zapewnienia szybkiej informacji zwrotnej jest ciągła integracja (patrz sekcja 1.2.4).

Gdy stosowane jest sekwencyjne podejście do wytwarzania, klient często nie widzi produktu, dopóki projekt nie zostanie prawie ukończony. W tym momencie często jest już za późno, aby zespół wytwórczy mógł skutecznie rozwiązać problemy użytkownika Uzyskując częste informacje zwrotne od klientów w miarę postępu projektu, zespół zwinny może włączyć większość nowych zmian do procesu wytwarzania produktu. Wczesne i częste informacje zwrotne pomagają zespołowi skupić się na funkcjach o największej wartości biznesowej lub powiązanym ryzyku, które są dostarczane klientowi w pierwszej kolejności. Pomaga to również lepiej zarządzać zespołem,

ponieważ możliwości zespołu są przejrzyste dla wszystkich. Np. ile pracy możemy wykonać w jednym sprincie lub jednej iteracji? Co przyspieszyłoby naszą pracę? Co uniemożliwia optymalizację pracy?

Korzyści płynące z wczesnej i częstej informacji zwrotnej obejmują:

- Unikanie nieporozumień związanych z wymaganiami, które mogły zostać wykryte dopiero w późniejszych etapach cyklu wytwórczego, kiedy ich naprawa jest bardziej kosztowna.
- Wyjaśnianie zgłoszeń klientów dotyczących funkcji i udostępnianie ich do użytku klientów na wczesnym etapie. W ten sposób produkt będzie lepiej odzwierciedlał oczekiwania klientów.
- Odkrywanie (poprzez ciągłą integrację), izolowanie i wczesne rozwiązywanie problemów jakościowych.
- Dostarczanie zespołowi zwinnemu informacji dotyczących jego wydajności i zdolności do realizacji zadań.
- Promowanie stałego tempa prac projektowych.

1.2 Aspekty podejść zwinnych

Istnieje kilka technik wytwarzania oprogramowania używanych przez organizacje twierdzące, iż używają metod zwinnych. Powszechne 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. Wspomniane wyżej techniki i praktyki zostaną omówione w tym podrozdziale.

1.2.1 Podejścia do zwinnego wytwarzania oprogramowania

Istnieje kilka podejść podejść do zwinnego wytwarzania oprogramowania, z których każde wdraża wartości i zasady Manifestu Agile na różne sposoby. 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: komunikację, prostotę, informacje zwrotne, odwagę i szacunek.

XP opisuje zbiór zasad jako dodatkowe wytyczne: zorientowanie na ludzi, ekonomia, wzajemne korzyści, samopodobieństwo, doskonalenie, różnorodność, refleksja, przepływ, możliwości, redundancja, niepowodzenie, jakość, małe kroki i przyjęta odpowiedzialność.

XP opisuje trzynaście podstawowych praktyk: dzielenie wspólnej przestrzeni, cały zespół, bogata w informacje przestrzeń robocza (ang. *informative workspace*), energiczna praca, programowanie w parach, historyjki, cykl tygodniowy, cykl kwartalny, luz, dziesięciominutowe budowanie wersji, ciągła integracja, programowanie w oparciu o zasadę „najpierw test" oraz o podejście przyrostowe.

Wiele używanych obecnie zwinnych podejść do tworzenia oprogramowania jest pod wpływem XP oraz jego wartości i zasad. Np. zespoły zwinne wykorzystujące Scrum często stosują praktyki XP.

Scrum

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

- Sprint: Scrum dzieli projekt na iteracje (zwane sprintami) o stałej długości (na ogół od dwóch do czterech tygodni).
- Przyrost produktu: wynikiem każdego sprintu jest produkt, który potencjalnie można wydać (nazywany przyrostem).
- Backlog produktu:właściciel produktu zarządza spriorytetyzowaną listą planowanych pozycji do zrobienia (nazywaną backlogiem produktu). Backlog produktu ewoluuje od sprintu do sprintu (nazywamy to udoskonalaniem backlogu produktu (ang. *backlog refinement*).
- 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 ten jest określany jako oparty na zasadzie „*pull*" (pobierania), a nie na zasadzie „*push*" („wpychania" pracy).
- Definicja ukończenia (ang. *Definition of Done*): by upewnić się, że na końcu każdego sprintu otrzymamy produkt potencjalnie nadający się do wdrożenia, zespół scrumowy omawia i definiuje 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 funkcje, które zespół chce ukończyć w ramach sprintu, są częścią rejestru sprintu. Jeżeli zespół wytwórczy nie jest w stanie ukończyć zadania w trakcie sprintu, powiązane funkcje produktu są usuwane ze sprintu, a zadanie wraca do backlogu produktu. Technika ta nazywa się ograniczaniem czasowym (ang. *timeboxing*).[3] Technikę tę stosuje się nie tylko do zadań, ale również w innych sytuacjach (np. egzekwowanie czasów rozpoczęcia i zakończenia spotkania).
- Przejrzystość: zespół wytwórczy codziennie raportuje i aktualizuje status sprintu na spotkaniu nazywanym codziennym Scrumem (ang. *daily Scrum*), dzięki czemu zawartość i postępy bieżącego sprintu, w tym wyniki testów, są dostępne dla zespołu scrumowego, kierownictwa i wszystkich zainteresowanych stron. Zespół wytwórczy może na przykład zaprezentować status sprintu na białej tablicy suchościeralnej.

[3] Brak tego fragmentu w oryginale, fragment dodany przez tłumaczy.

Scrum definiuje trzy role członków zespołu scrumowego:

- Scrum Master zapewnia, że praktyki i reguły scrumowe są wdrażane i odpowiednio stosowane, usuwa ich naruszenia, rozwiązuje problemy z zasobami i usuwa inne przeszkody, które mogłyby uniemożliwić zespołowi stosowanie praktyk i zasad. Scrum Master nie jest liderem zespołu, ale trenerem (coachem).
- Właściciel produktu reprezentuje przyszłego klienta oraz tworzy, zarządza i ustala priorytety w backlogu produktu. Właściciel produktu nie jest liderem zespołu.
- Zespół wytwórczy składa się z trzech do dziewięciu osób,[4] 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 narzuca konkretnych technik wytwarzania oprogramowania (np. zasada „najpierw test"), które powinny być stosowane. Co więcej, Scrum nie zawiera wytycznych dotyczących sposobu przeprowadzenia testów w projekcie scrumowym.

Kanban

Kanban [Anderson13] to podejście do zarządzania, które jest czasami stosowane w projektach zwinnych. Ogólnym celem jest wizualizacja i optymalizacja przepływu pracy w łańcuchu wartości dodanej (ang. *a value-added chain*). Kanban wykorzystuje trzy narzędzia [Linz14]:

- Tablica Kanban: łańcuch wartości, którym należy zarządzać, jest wizualizowany na tablicy Kanban. Każda kolumna przedstawia 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 wytwarzania.
- 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 dla 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 poprzez 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, które nie zostały jeszcze zaplanowane, czekają w rejestrze produktu (ang. *product backlog*) i są przenoszone na tablicę Kanban; w momencie, gdy zwalnia się miejsce, w kolejnym etapie są one odpowiednio przesuwane na tablicy kanbanowej.

[4] Brak tego fragmentu w oryginale, fragment dodany przez tłumaczy.

Iteracje i sprinty są opcjonalne w Kanbanie. Proces Kanban pozwala na wydawanie produktów jeden po drugim (element po elemencie), a nie jako część wydania. Ograniczenia czasowe (ang. *timeboxing*) jako mechanizm synchronizacji jest opcjonalny, w przeciwieństwie do Scruma, który synchronizuje wszystkie zadania w ramach sprintu.

1.2.2 Wspólne tworzenie historyjek użytkownika

Słabej jakości specyfikacje są często główną przyczyną niepowodzeń projektu. Problemy ze specyfikacją mogą wynikać z braku zrozumienia przez użytkownika własnych potrzeb, braku globalnej wizji systemu, nadmiarowych lub sprzecznych funkcji oraz innych nieporozumień. W środowisku zwinnym historyjki użytkownika pisane są w celu uchwycenia wymagań z perspektywy programistów, testerów i przedstawicieli jednostek biznesowych. W przeciwieństwie do sekwencyjnego cyklu wytwarzania oprogramowania, gdzie wspólna wizja funkcji jest realizowana poprzez przeglądy formalne po napisaniu wymagań, w zwinnym wytwarzaniu oprogramowania ta wspólna wizja jest realizowana przez częste przeglądy nieformalne podczas pisania wymagań.

Historyjki użytkownika muszą odnosić się zarówno do cech funkcjonalnych, jak i niefunkcjonalnych. Każda historyjka zawiera kryteria akceptacji dla tych cech. Kryteria te powinny zostać zdefiniowane we współpracy pomiędzy przedstawicielami jednostek biznesowych, programistami i testerami. Kryteria te dostarczają programistom i testerom rozszerzoną wizję funkcji, którą reprezentanci biznesu będą weryfikować. Zespół zwinny uznaje zadanie za ukończone, gdy zestaw kryteriów akceptacji zostanie spełniony.

Zazwyczaj unikalna perspektywa testera pozwala ulepszyć historyjkę użytkownika poprzez zidentyfikowanie brakujących szczegółów lub wymagań niefunkcjonalnych. Wkład testera może przyjmować formę zadawania przedstawicielom jednostek biznesowych otwartych pytań dotyczących historyjki użytkownika, proponowania sposobów jej przetestowania oraz potwierdzania spełnienia kryteriów akceptacji.

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

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

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

- Karta (ang. *Card*): karta jest fizycznym nośnikiem opisującym historyjkę użytkownika. Identyfikuje wymaganie, jego krytyczność, oczekiwany czas wykonania i testowania oraz kryteria akceptacji dla tej historyjki. Opis powinien być precyzyjny, gdyż będzie wykorzystany w backlogu produktu.
- Konwersacja (ang. *Conversation*): konwersacja wyjaśnia, w jaki sposób oprogramowanie będzie używane. Może to być udokumentowane lub przekazane ustnie. Testerzy, mający inny punkt widzenia niż programiści 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 obejmować wiele historyjek użytkownika. Do sprawdzenia spełnienia kryteriów powinny zostać użyte testy zarówno pozytywne, jak i negatywne. Podczas potwierdzania różni uczestnicy odgrywają rolę testerów. Mogą to być zarówno programiści, jak i specjaliści od wydajności, bezpieczeństwa, współdziałania i innych charakterystyk jakościowych. Aby potwierdzić, że historyjka może zostać zamknięta, należy przetestować zdefiniowane kryteria akceptacji i wykazać, że są one spełnione.

Zespoły zwinne różnią się pod względem sposobu dokumentowania historyjek użytkownika. Niezależnie od przyjętego podejścia, dokumentacja powinna być zwięzła, wystarczająca i ograniczać się do niezbędnych elementów.

1.2.3 Retrospektywy

W wytwarzaniu zwinnym retrospektywa to spotkanie odbywające się na końcu każdej iteracji w celu omówienia tego, co zakończyło się sukcesem, co można poprawić, jak wdrożyć poprawki i powtórnie osiągnąć sukces w przyszłych iteracjach. Retrospektywy obejmują takie zagadnienia jak: proces, ludzie, organizacja, relacje i narzędzia. Regularnie przeprowadzane retrospektywy, podczas których podejmowane są odpowiednie działania, mają kluczowe znaczenie dla samoorganizacji i ciągłego doskonalenia procesów wytwarzania oraz testowania.

Retrospektywy mogą skutkować decyzjami dotyczącymi doskonalenia testów, koncentrującymi się na skuteczności testów, produktywności testów, jakości przypadków testowych oraz satysfakcji zespołu. Retrospektywy mogą również dotyczyć testowalności aplikacji, historyjek użytkownika, cech systemu lub jego interfejsów. Ponadto analiza podstawowych przyczyn defektów przeprowadzona podczas retrospektywy może przyczynić się do usprawnień w zakresie testowania oraz wytwarzania. Na ogół przyjmuje się, iż zespoły powinny wdrażać tylko kilka usprawnień podczas jednej iteracji. Pozwala to na ciągłe doskonalenie w stałym tempie.

Czas i organizacja retrospektyw zależy od stosowanej metodyki zwinnej. Przedstawiciele jednostek biznesowych oraz zespół są uczestnikami retrospektyw, a moderator

organizuje i prowadzi spotkania. W niektórych przypadkach zespoły mogą zaprosić na spotkanie innych uczestników.

Testerzy powinni odgrywać ważną rolę w retrospektywach. Testerzy są częścią zespołu i wnoszą swoją unikalną perspektywę [ISTQB_FL_SYL, rozdz. 1.5]. Testowanie jest wykonywane w każdym sprincie i w istotny sposób przyczynia się do sukcesu każdego sprintu. Wszyscy członkowie zespołu, testerzy i osoby nie będące testerami, mogą wnieść swój 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 cechy innych przeglądów [ISTQB_FL_SYL, podrozdział 3.2].

1.2.4 Ciągła integracja

Dostarczenie przyrostu produktu wymaga niezawodnego, działającego, zintegrowanego oprogramowania na koniec każdego sprintu. Ciągła integracja pozwala sprostać temu wyzwaniu poprzez regularne scalanie wszystkich zmian wprowadzonych w oprogramowaniu i regularną integrację wszystkich zmienionych modułów przynajmniej raz dziennie. Zarządzanie konfiguracją, kompilacja, budowanie wersji, wdrażanie i testowanie są połączone w jeden zautomatyzowany, powtarzalny proces. Ponieważ programiści stale integrują swoją pracę, budują i stale testują, defekty w kodzie są wykrywane szybciej.

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

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

Zautomatyzowany proces budowania i testowania odbywa się codziennie i pozwala na wczesne i szybkie wykrywanie błędów integracji. Ciągła integracja pozwala testerom w projekcie zwinnym na regularne uruchamianie testów automatycznych (w niektórych przypadkach w ramach samego procesu ciągłej integracji) i szybkie przekazywanie zespołowi informacji zwrotnej na temat jakości kodu. Te wyniki testów są dostępne dla każdego członka zespołu, zwłaszcza gdy zautomatyzowane raporty są zintegrowane z procesem. Zautomatyzowane testy regresji mogą być wykonywane w sposób ciągły

przez całą iterację. Dobre zautomatyzowane testy regresji obejmują jak najwięcej funkcjonalności, włącznie z historyjkami użytkownika dostarczonymi w poprzednich iteracjach. Dobre pokrycie zautomatyzowanymi testami regresji wspiera budowanie (i testowanie) dużych, zintegrowanych systemów. Gdy testy regresji są zautomatyzowane, testerzy zwinni mogą się skoncentrować na manualnych testach nowych funkcji, zaimplementowanych zmianach oraz testach potwierdzających naprawienie defektów.

Organizacje stosujące ciągłą integrację jako uzupełnienie testów automatycznych na ogół używają narzędzi do budowy wersji (ang. *build tools*) do wdrożenia ciągłej kontroli jakości. Poza wykonywaniem testów modułowych (jednostkowych) i integracyjnych, narzędzia te mogą wykonywać dodatkowo testy statyczne i dynamiczne, mierzyć i profilować wydajność, wyodrębniać i formatować dokumentację z kodu źródłowego, a także ułatwiać manualne procesy zapewniania jakości. To nieustanne stosowanie kontroli jakości ma na celu zarówno podnoszenie jakości produktu, jak również skrócenie czasu potrzebnego na jego dostarczenie poprzez zastąpienie tradycyjnej praktyki stosowania kontroli jakości po zakończeniu całego procesu wytwarzania produktu.

Narzędzia do budowy wersji można połączyć z narzędziami do automatycznego wdrażania (ang. *automatic deployment tool*), które mogą przenieść odpowiednią wersję z serwera ciągłej integracji i budowy i wdrożyć bezpośrednio do jednego lub kilku środowisk deweloperskich, testowych, preprodukcyjnych, a nawet produkcyjnych. Zmniejsza to liczbę błędów i opóźnień związanych z poleganiem na wyspecjalizowanej kadrze lub programistach instalujących wydania w tych środowiskach.

Ciągła integracja może przynieść następujące korzyści:

- Umożliwia wcześniejsze wykrywanie i łatwiejszą analizę podstawowych przyczyn problemów z integracją oraz zmian powodujących konflikty.
- Umożliwia przekazywanie zespołowi wytwórczemu regularnych informacji zwrotnych na temat tego, czy kod działa.
- Umożliwia utrzymywanie wersji testowanego oprogramowania: testerzy zwinni są co najwyżej jeden dzień za deweloperską wersją oprogramowania.
- Zmniejsza ryzyko regresji związane z refaktoryzacją kodu napisanego przez deweloperów dzięki szybkiemu ponownemu testowaniu bazy kodu po każdym małym zestawie zmian.
- Daje pewność, że codzienna praca nad rozwojem jest oparta na solidnych podstawach.
- Zapewnia, że postęp w kierunku ukończenia przyrostu produktu jest widoczny, jednocześnie motywując deweloperów i testerów.
- Eliminuje ryzyko opóźnień w projekcie występujące w przypadku integracji typu „wielki wybuch”.
- Zapewnia stałą dostępność wykonywalnego oprogramowania przez cały czas trwania sprintu do testów, celów demonstracyjnych lub szkoleniowych.
- Umożliwia ograniczenie powtarzalnych testów manualnych.
- Umożliwia szybką informację zwrotną na temat decyzji podjętych w celu

doskonalenia jakości i testów.

Ciągła integracja nie jest jednak pozbawiona ryzyka i wyzwań:

- Należy wdrożyć i utrzymywać narzędzia ciągłej integracji.
- Proces ciągłej integracji musi zostać zdefiniowany i ustanowiony.
- Automatyzacja testów wymaga dodatkowych zasobów, a jej ustanowienie może być skomplikowane.
- Dokładne pokrycie testami jest niezbędne do osiągnięcia korzyści z testowania automatycznego.
- Zespoły czasami nadmiernie polegają na testach modułowych, 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 [ISTQB_FL_SYL], planowanie jest działaniem ciągłym i tak jest również w przypadku zwinnego cyklu życia. W zwinnym cyklu życia występują dwa rodzaje planowania: planowanie wydania i planowanie iteracji.

Planowanie wydania dotyczy wydanie produktu, często na kilka miesięcy przed rozpoczęciem projektu. Planowanie wydania definiuje i redefiniuje backlog produktu i może obejmować udoskonalenie większych historyjek użytkownika poprzez przekształcanie ich w zbiór mniejszych historyjek. Planowanie wydania stanowi podstawę podejścia do testów oraz planu testów obejmującego wszystkie iteracje. Plany wydania są wysokopoziomowe.

W planowaniu wydania przedstawiciele biznesu ustalają i priorytetyzują historyjki użytkownika dla wydania, we współpracy z zespołem (patrz sekcja 1.2.2). W oparciu o te historyjki określane są ryzyka projektowe i jakościowe oraz wykonywane jest wysokopoziomowe szacowanie nakładu pracy (patrz podrozdział 3.2).

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

- Definiowanie testowalnych historyjek użytkownika, włączając w to kryteria akceptacji.
- Udział w analizie ryzyka projektowego i jakościowego.
- Szacowanie nakładu pracy związanego z testowaniem historyjek użytkownika.
- Definiowanie potrzebnych poziomów testów.
- Planowanie testów dla wydania.

Po zakończeniu planowania wydania rozpoczyna się planowanie iteracji dla pierwszej iteracji. Planowaniu iteracji wybiega w przyszłość tylko do końca pojedynczej iteracji, koncentrując się na backlogu iteracji.

Podczas planowania iteracji zespół wybiera historyjki użytkownika z grupy historyjek ustawionych według priorytetu w backlogu produktu, uszczegóławia historyjki użytkownika, przeprowadza analizę ryzyka dla historyjek oraz szacuje pracę potrzebną do wykonania każdej historyjki. Jeżeli jakaś historyjka użytkownika jest zbyt niejasna (nieprecyzyjna), a próby jej wyjaśnienia nie powiodły się, zespół może odrzucić tę historyjkę i wziąć następną historyjkę zgodnie z priorytetami. Przedstawiciele biznesu mają obowiązek odpowiadać na pytania zespołu dotyczące każdej historyjki, aby zespół mógł zrozumieć, jak zaimplementować każdą historyjkę i jak ją przetestować.

Liczba wybranych historyjek zależy od zakładanego tempa pracy zespołu (ang. *team velocity*) i oszacowanego rozmiaru wybranych historyjek. Gdy zawartość iteracji zostanie uzgodniona, historyjki dzielone są na zadania, które będą wykonywane przez poszczególnych 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.
- Podział 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 testów.

Plany wydania mogą się zmieniać w miarę postępu projektu, włączając w to zmiany w poszczególnych historyjkach użytkownika w backlogu produktu. Zmiany te mogą być wywołane przez czynniki wewnętrzne lub zewnętrzne. Czynniki wewnętrzne obejmują możliwości dostarczania, prędkość oraz problemy techniczne. Czynniki zewnętrzne obejmują nowe rynki i możliwości, nowych konkurentów lub zagrożenia biznesowe, które mogą zmienić cele wydania i/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.

Zmiany te mogą stanowić wyzwanie dla testerów. Testerzy muszą rozumieć ogólny obraz wydania do celów planowania testów. Dodatkowo muszą mieć adekwatną podstawę testów i wyrocznię testową w każdej iteracji dla celów tworzenia testów, jak omówiono w [ISTQB_FL_SYL, podrozdział 1.4]. Wymagane informacje muszą być dostępne dla testera na wczesnym etapie, a jednocześnie zmiany muszą być wprowadzane zgodnie z metodykami zwinnymi. Dylemat ten wymaga podejmowania ostrożnych decyzji dotyczących strategii testów i dokumentacji testowej. Więcej informacji na temat wyzwań związanych z testowaniem zwinnym można znaleźć w [Black09, rozdział 12].

Planowanie wydań oraz iteracji powinno obejmować również planowanie testów, a także planowanie działań wytwórczych. Szczegółowe kwestie związane z testami, którymi

należy się zająć, obejmują:

- Zakres i rozmiar testów dla obszarów objętych zakresem, cele testów wraz z uzasadnieniem tych decyzji.
- Członków zespołu, którzy będą wykonywali zadania testowe.
- Potrzebne środowiska testowe i dane testowe, kiedy są one potrzebne oraz czy będą wymagane przed lub w trakcie projektu jakieś uzupełnienia lub zmiany w środowisku testowym i danych testowych.
- Harmonogram, kolejność, zależności i warunki wstępne dla czynności testowania funkcjonalnego i niefunkcjonalnego (np. jak często należy wykonywać testy regresji, które funkcje zależą od innych funkcji lub danych testowych itp.), włączając w to informacje, w jaki sposób czynności testowe zależą od zadań deweloperskich.
- Ryzyka projektowe i jakościowe, którymi należy się zająć (patrz sekcja 3.2.1).

Ponadto, większy zespół szacujący pracochłonność powinien uwzględnić czas i pracochłonność potrzebne 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 a 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 projektach zwinnych.

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 narzędzia i techniki stosowane do informowania o statusie testów w projekcie zwinnym, w tym postępu testów i jakości produktu.

FA-2.2.2 (K2) Kandydat potrafi opisać proces ewoluowania testów w wielu iteracjach i wyjaśnić, dlaczego automatyzacja testów jest istotna dla zarządzania 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 zespole zwinnym.

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

2.1 Różnice pomiędzy tradycyjnym a 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 różni się w różnych cyklach życia oprogramowania. Testerzy muszą rozumieć różnice pomiędzy testowaniem w tradycyjnych modelach cyklu życia (np. sekwencyjnym jak model V czy iteracyjnym jak RUP) a testowaniem w 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, produktami pracy projektowej, stosowanymi nazwami, kryteriami wejścia i wyjścia stosowanymi na różnych poziomach testów, wykorzystaniem narzędzi oraz sposobem, w jaki niezależne testowanie może być skutecznie wykorzystane.

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

2.1.1 Testowanie i czynności wytwórcze

Jedną z podstawowych różnic pomiędzy tradycyjnymi a zwinnymi cyklami życia jest idea bardzo krótkich iteracji, z których każda skutkuje działającym oprogramowaniem dostarczającym interesariuszom biznesowym wartościową funkcjonalność. Na początku projektu następuje okres planowania wydania. Następnie mamy sekwencję iteracji. Na początku każdej iteracji występuje okres jej planowania. Po ustaleniu zakresu iteracji, wybrane historyjki użytkownika są opracowywane, integrowane z systemem oraz testowane. Tego typu iteracje są wysoce dynamiczne, a czynnościami wytwarzania, integracji i testowania odbywają się w trakcie każdej iteracji, przy czym są one w znaczym stopniu równoległe i nakładają się na siebie. Czynności testowe wykonywane są przez całą iterację (np. codziennie), a nie tylko jako końcowe czynności iteracji.

Testerzy, deweloperzy i interesariusze biznesowi odgrywają rolę w testowaniu, podobnie jak w przypadku tradycyjnych cykli życia. Deweloperzy wykonują testy modułowe podczas opracowywania cech funkcjonalnych na podstawie historyjek użytkownika. Następnie testerzy testują te cechy funkcjonalne. Właściciele produktu (przedstawiciele biznesu) także testują historyjki podczas ich wdrażania. Interesariusze biznesowi mogą wykorzystywać napisane przypadki testowe, ale mogą także eksperymentować z funkcjami i używać ich, by zapewnić szybką informację zwrotną zespołowi wytwórczemu.

W niektórych przypadkach okresowo wykonywane są iteracje „scalające” i stabilizujące, mające na celu rozwiązanie utrzymujących się defektów lub innych form długu technicznego. Tym niemniej, najlepszą praktyką jest nieuznawanie funkcji 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 część zadań sprintu dla tej iteracji (nazywamy to podejściem „najpierw napraw usterki" (ang. *fix bugs first*)). Tym niemniej niektórzy twierdzą, że praktyka ta prowadzi do sytuacji, w której całkowita praca do wykonania w danej iteracji jest nieznana i trudniej będzie oszacować, kiedy pozostałe funkcje mogą być zakończone. Na końcu sekwencji iteracji może wystąpić zestaw czynności związanych z wydaniem, aby przygotować oprogramowanie do dostarczenia, chociaż w niektórych przypadkach dostawa następuje na końcu każdej iteracji.

Gdy jako jedną ze strategii testowych stosuje się testowanie oparte na ryzyku, 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 funkcji. Wpływa także na oszacowanie wymaganego nakładu pracy (np. liczby „punktów historyjki") dla każdej cechy funkcjonalnej.

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

Testerzy mogą także pracować w ramach zespołu jako trenerzy testowania i jakości (ang. *coach*), dzieląc się przy tym wiedzą związaną z testowaniem i wspierając prace związane z zapewnieniem jakości w zespole. Promuje to poczucie wspólnej odpowiedzialności za jakość produktu.

W wielu zespołach zwinnych automatyzacja testów ma miejsce na wszystkich poziomach testów. Może to oznaczać, że testerzy spędzają więcej czasu na tworzeniu, monitorowaniu oraz zarządzaniu testami automatycznymi i uzyskanymi wynikami. Ze względu na intensywne wykorzystanie automatyzacji testów coraz większy odsetek testów manualnych w projektach zwinnych jest wykonywany z użyciem technik opartych na defektach oraz na doświadczeniu, takich jak ataki na oprogramowanie, testowanie eksploracyjne czy też zgadywanie błędów (patrz [ISTQB_ALTA_SYL, podrozdziały 3.3 i 3.4] oraz [ISTQB_FL_SYL, podrozdział 4.5]). Testerzy w projekcie zwinnym powinni być też gotowi do udziału w tworzeniu i wykonywaniu testów automatycznych. Podczas gdy deweloperzy będą się skupiać na tworzeniu testów modułowych, testerzy powinni się koncentrować na tworzeniu zautomatyzowanych testów integracyjnych, testów systemowych oraz testach integracji systemu. Prowadzi to do tendencji faworyzowania w zespole zwinnym testerów o mocnych podstawach technicznych i umiejętnościach w zakresie automatyzacji testów.

Jedną z podstawowych zasad zwinności jest to, że zmiany są dozwolone przez cały czas trwania projektu. Dlatego też w projektach zwinnych preferowana jest tzw. „lekka

dokumentacja projektowa". Zmiany istniejących funkcji mają wpływ na testy, szczególnie na testy regresji. Wykorzystanie testów automatycznych jest jedną z metod zarządzania nakładem pracy na testy związane ze zmianami. Tym niemniej istotne jest, by zakres i tempo zmian nie przekraczały zdolności zespołu projektowego do radzenia sobie z ryzykami związanymi z tymi zmianami.

2.1.2 Projektowe produkty pracy

Projektowe produkty pracy dzielą się na trzy kategorie:

1. Produkty pracy zorientowane na biznes opisujące, co jest niezbędne (np. specyfikacje wymagań) i jak z tego korzystać (np. dokumentacja użytkownika).
2. Produkty prac wytwórczych opisujące, jak system jest zbudowany (np. diagramy związków encji bazy danych), jak zaimplementowano system (np. kod) oraz jak sprawdzano poszczególne fragmenty kodu (np. automatyczne testy modułowe).
3. Produkty prac testowych, które opisują sposób testowania systemu (np. strategie i plany testów), które faktycznie testują system (np. testy automatyczne i manualne) lub które przedstawiają wyniki testów (np. tablica wskaźników (ang. *dashboard*), co omówiono w sekcji 2.2.1).

W typowym projekcie zwinnym nacisk kładziony jest 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 ograniczenia dokumentacji stosuje się tylko do dokumentacji, która nie stanowi wartości dla klienta. W udanej implementacji zwinnej utrzymana jest równowaga pomiędzy – z jednej strony – rosnącym naciskiem na redukowanie dokumentacji a – 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 pracy są wymagane i jaki jest wymagany poziom szczegółowości dokumentacji produktu pracy.

Typowe, zorientowane na klienta produkty pracy w projekcie zwinnym to historyjki użytkownika i kryteria akceptacji. Historyjki użytkownika w projekcie zwinnym są odpowiednikiem specyfikacji wymagań użytkownika. Powinny wyjaśniać, jak system powinien się zachowywać w odniesieniu do pojedynczej, spójnej cechy lub funkcji. Historyjka użytkownika powinna definiować funkcję na tyle małą, by można ją było ukończyć w jednej iteracji. Większe zbiory powiązanych funkcji lub zbiór podfunkcji sumujących się w pojedynczą bardziej skomplikowaną funkcję, są nazywane opowieściami (ang. *epic*). Opowieści mogą obejmować historyjki użytkownika dla różnych zespołów wytwórczych. Na przykład pojedyncza historyjka użytkownika może opisywać to, co jest wymagane na poziomie API (warstwa pośrednicząca (ang. *middleware*)), a inna historyjka może opisywać to, co jest potrzebne na poziomie interfejsu użytkownika (aplikacja). Zbiory te mogą być opracowywane w serii sprintów. Każda z opowieści oraz zawarte w niej historyjki użytkownika mają zdefiniowane kryteria akceptacji ustalające, kiedy można uznać je za zakończone.

Typowe produkty pracy deweloperów w projektach zwinnych zawierają kod. Tym niemniej deweloperzy pracujący w projektach zwinnych często tworzą automatyczne testy modułowe (jednostkowe). Testy te mogą powstawać po napisaniu kodu. Jednak w niektórych przypadkach programiści tworzą te testy przyrostowo przed napisaniem nowego fragmentu kodu tak, aby zapewnić sposób weryfikacji po napisaniu każdej części kodu, czy działa ona zgodnie z oczekiwaniami. Podczas gdy podejście to jest określane jako „najpierw test" lub wytwarzanie sterowane testami, w rzeczywistości testy są raczej formą wykonywalnych niskopoziomowych specyfikacji projektowych, a nie testów [Beck02].

Typowe produkty prac testowych w projektach zwinnych to testy automatyczne oraz takie dokumenty jak: plany testów, katalogi ryzyk jakościowych, testy manualne, raporty o defektach oraz dzienniki (logi) wyników testów. Dokumenty te tworzone 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 tworzyć metryki testowe z raportów o defektach i dzienników (logów) wyników testów. Tutaj również nacisk kładziony jest na lekkość podejścia.

W pewnych wdrożeniach zwinnych, zwłaszcza w przypadku projektów i produktów podlegających regulacjom prawnym, krytycznych ze względów bezpieczeństwa, rozproszonych lub wysoce złożonych, wymagana jest dalsza formalizacja tych produktów pracy. Na przykład niektóre zespoły przekształcają historyjki użytkownika i kryteria akceptacji w bardziej formalne specyfikacje wymagań. Można przygotować raporty śledzenia pionowego i poziomego (ang. *vertical and horizontal traceability reports*), by spełnić wymagania audytorów, wymogi prawne oraz inne wymagania.

2.1.3 Poziomy testów

Poziomy testów to czynności testowe, które są ze sobą logicznie powiązane, często poprzez dojrzałość lub kompletność testowanych elementów.

W sekwencyjnych modelach cyklu życia poziomy testów są często definiowane w taki sposób, że kryteria wyjścia z jednego poziomu są częścią kryteriów wejścia dla następnego poziomu. W niektórych modelach iteracyjnych ta reguła nie ma zastosowania. Poziomy testów nakładają się na siebie. Specyfikacja wymagań, specyfikacja projektowa oraz czynności wytwórcze mogą pokrywać 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 zaplanowaniu iteracji, to w praktyce można takie zmiany zaobserwować. Podczas iteracji każda historyjka użytkownika zazwyczaj przechodzi kolejno przez następujące czynności testowe:

- testy jednostkowe (modułowe) na ogół wykonywane przez deweloperów,
- testy akceptacyjne cechy (funkcji), które czasami dzielone są na dwie

czynności:
- testy weryfikacyjne funkcji, które są często automatyzowane; mogą być wykonywane przez deweloperów lub testerów i obejmują testowanie kryteriów akceptacji historyjek użytkownika,
- testy walidacyjne funkcji, które są zwykle manualne; mogą być wykonywane przez deweloperów, testerów i interesariuszy biznesowych pracujących razem w celu ustalenia, 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 przeprowadzania testów regresji. Obejmuje to ponowne uruchamianie zautomatyzowanych testów modułowych (jednostkowych) i testów weryfikacji funkcji z bieżącej i poprzednich iteracji, zwykle z wykorzystaniem narzędzi do ciągłej integracji (ang. *continuous integration framework*).

W projektach zwinnych może istnieć także poziom testów systemowych, który rozpoczyna się, gdy pierwsza historyjka użytkownika jest gotowa do takich testów. Ów poziom może obejmować zarówno testy funkcjonalne, jak i niefunkcjonalne testy wydajności, niezawodności, użyteczności oraz inne istotne typy testów.

Zespoły zwinne mogą stosować różne formy testów akceptacyjnych (zgodnie z terminologią zawartą w [ISTQB_FL_SYL]). Wewnętrzne testy alfa oraz zewnętrzne testy beta mogą być przeprowadzane na zakończenie każdej iteracji, po zakończeniu każdej iteracji lub po serii kilku iteracji. Testy akceptacyjne użytkownika, operacyjne testy akceptacyjne, testy akceptacyjne zgodności z umową i testy akceptacyjne zgodności z prawem mogą mieć miejsce na zakończenie każdej iteracji, po zakończeniu każdej iteracji lub po serii iteracji.

2.1.4 Testowanie i zarządzanie konfiguracją

Projekty zwinne często wymagają intensywnego wykorzystywania automatycznych narzędzi do opracowywania, testowania oraz zarządzania rozwojem oprogramowania. Programiści wykorzystują narzędzia do analizy statycznej, testów modułowych (jednostkowych) i do mierzenia pokrycia kodu. Programiści ciągle komitują kod i testy modułowe do systemów zarządzania konfiguracją przy użyciu struktur budowania i testowania. Takie struktury umożliwiają ciągłą integrację tworzonego oprogramowania z systemem; analiza statyczna i testy modułowe są uruchamiane wielokrotnie, gdy tylko nowe oprogramowanie zostanie wprowadzone.

Testy automatyczne mogą również obejmować testy funkcjonalne na poziomach integracyjnym i systemowym. Takie zautomatyzowane 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 modułowych i wykonywane rzadziej. Na przykład testy modułowe mogą być uruchamiane za każdym razem, gdy wprowadzane jest nowe oprogramowanie, podczas gdy dłuższe testy funkcjonalne są

uruchamiane tylko co kilka dni.

Jednym z celów testów automatycznych jest potwierdzenie, że testowana wersja działa i można ją zainstalować. Jeżeli którykolwiek test automatyczny nie został zaliczony, zespół powinien naprawić wykrytą usterkę do czasu następnego komitowania kodu. Wymaga to zainwestowania w raportowanie testów działające w czasie rzeczywistym, by móc zapewnić dobry wgląd w wyniki testów. Tego typu podejście pomaga zredukować kosztowne i nieefektywne cykle „zbuduj–zainstaluj–awaria–przebuduj–ponownie zainstaluj", które mogą wystąpić w wielu tradycyjnych projektach, ponieważ zmiany, które niszczą wersję lub powodują, że oprogramowanie nie może zostać zainstalowane, są szybko wykonywane.

Inną korzyścią szerokiej automatyzacji testów i wykorzystania narzędzi do budowy wersji i testowania jest to, że wspomaga ona zarządzanie ryzykiem regresji związanym z częstymi zmianami, które występują w projektach zwinnych. Jednakże nadmierne poleganie jedynie na automatycznych testach modułowych (jednostrkowych) w celu zarządzania tym ryzykiem może stanowić problem, bowiem testowanie modułowe ma ograniczoną skuteczność wykrywania defektów [Jones11]. Automatyczne testy wymagane są także na poziomach integracyjnym i systemowym.

2.1.5 Możliwości organizacyjne niezależnego testowania

Jak omówiono w [ISTQB_FL_SYL], niezależni testerzy są często bardziej efektywni w znajdowaniu defektów. W niektórych zespołach zwinnych deweloperzy wytwarzają większość testów w postaci 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 presja czasu, brak zrozumienia nowych cech funkcjonalnych produktu oraz problemy w relacjach z interesariuszami biznesowymi i deweloperami mogą często powodować problemy przy takim podejściu.

Trzecią opcją jest posiadanie niezależnego, oddzielonego zespołu testowego, w którym testerzy są przydzieleni do zespołu zwinnego w perspektywie długoterminowej na początku projektu, co umożliwia im zachowanie niezależności przy jednoczesnym uzyskaniu dobrego zrozumienia produktu i nawiązaniu silnych relacji z innymi członkami zespołu. Ponadto, 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 testami na tych poziomach, które trudno wpasować w sprinty (np. testy integracji systemów).

2.2 Status testowania w projektach zwinnych

W projektach zwinnych zmiany zachodzą bardzo szybko. Zmiany te oznaczają, że zarówno status oraz postęp testów, jak i jakość produktu, mogą ewoluować 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 te mogą wpływać na istniejące już cechy funkcjonalne wytworzone w poprzednich iteracjach. Dlatego też testy manualne i automatyczne muszą być aktualizowane, aby skutecznie 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, kiedy zespół będzie miał działające oprogramowanie, musi on monitorować postęp wszystkich elementów pracy w trakcie iteracji i wydania. Testerzy w zespołach zwinnych wykorzystują różne metody rejestrowania postępu i statusu testów, w tym wyniki testów automatycznych, postęp w zadaniach testowych i historyjkach na tablicy zadań zwinnych oraz wykresy spalania (ang. *burndown charts*) pokazujące postęp prac zespołu. Informacje te mogą być następnie komunikowane reszcie zespołu przy użyciu takich środków jak tablica wskaźników w wiki, wiadomości e-mail w stylu tablicy wskaźników, a także werbalnie podczas codziennych spotkań na stojąco (ang. *stand-up meeting*). Zespoły zwinne mogą używać narzędzi, które automatycznie generują raporty o statusie w oparciu o wyniki testów i postęp prac, które z kolei aktualizują tablicę wskaźników oraz wiadomości e-mail w stylu wiki. Ta metoda komunikacji gromadzi także metryki z procesu testowania, które można wykorzystać do doskonalenia procesu. Komunikowanie statusu testów w taki zautomatyzowany 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 stałą, szczegółową wizualną reprezentację 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 rejestrowane 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 między kolumnami takimi jak: „do zrobienia”, „w toku”, „weryfikacja” oraz „zrobione”. Zespoły zwinne mogą używać narzędzi do obsługi kart historyjekoraz tablicy zadań, które to narzędzia mogą automatyzować uaktualnianie tablicy wskaźników i statusu.

Zadania testowe na tablicy zadań są powiązane z kryteriami akceptacji zdefiniowanymi dla każdej historyjki użytkownika. Gdy skrypty testów automatycznych, testy manualne i eksploracyjne dla zadania testowego osiągną status „wykonane", zadanie jest przenoszone na tablicy zadań do kolumny „zrobione". Cały zespół regularnie przegląda status zadań na tablicy, często podczas codziennych spotkań na stojąco (ang. *daily stand up meetings*), 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 problemy blokujące postęp zadania.

Codzienne spotkanie na stojąco angażuje wszystkich członków zespołu zwinnego, także testerów. Podczas tego spotkania podają oni aktualny status swoich zadań. Agenda dla każdego członka zespołu to [Agile Alliance Guide]:

- Co skończyłeś/aś 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 na stojąco, więc cały zespół jest ich świadomy i rozwiązuje je razem.

Aby poprawić ogólną jakość produktu, wiele zespołów zwinnych przeprowadza badania satysfakcji klientów, aby otrzymać informacje zwrotne na temat tego, na ile produkt spełnia oczekiwania klientów. W celu poprawy jakości zespoły mogą używać metryk podobnych do tych wykorzystywanych w tradycyjnych metodykach wytwarzania takich jak wskaźniki „zaliczenia/niezaliczenia" testów, wskaźniki wykrywania defektów, wyniki testów potwierdzających i regresji, gęstość defektów, usterki znalezione i naprawione, pokrycie wymagań, pokrycie ryzyka, pokrycie kodu oraz modyfikacje kodu. Podobnie jak w przypadku każdego cyklu życia, rejestrowane i raportowane metryki powinny być odpowiednie i powinny ułatwiać podejmowanie decyzji. Metryki nie powinny być wykorzystywane do nagradzania, karania lub izolowania 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 wprowadzonych w czasie bieżącej iteracji, testerzy sprawdzają także, czy nie obniżono jakości cech funkcjonalnych (sprawdzenie pod kątem regresji), które zostały wytworzone i przetestowane w poprzednich iteracjach. Ryzyko wprowadzenia regresji do funkcji 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 funkcje. By utrzymać prędkość wytwarzania bez zaciągania dużego długu technicznego, krytyczna staje się inwestycja zespołu w automatyzację testów na wszystkich poziomach testów tak wcześnie, jak to możliwe. Krytyczne jest także, by wszystkie zasoby testowe takie jak: testy automatyczne, przypadki testowe dla testów manualnych, dane testowe i inne

artefakty testowe były aktualizowane w każdej iteracji. Dlatego też zaleca się, aby wszystkie zasoby testowe były zarządzane przy użyciu narzędzia do zarządzania konfiguracją. Zapewnia to kontrolę wersji, ułatwia dostęp do zasobów testowych wszystkim członkom zespołu, ułatwia zmiany wymagane przez zmieniającą się funkcjonalność, a równocześnie ciągle zachowuje historyczne informacje o zasobach testowych.

Ponieważ całkowite powtórzenie wszystkich testów rzadko jest możliwe, zwłaszcza w projektach zwinnych o napiętym harmonogramie, testerzy powinni przeznaczyć czas w każdej iteracji na przegląd manualnych i zautomatyzowanych przypadków testowych z poprzednich i z aktualnej iteracji, by wybrać przypadki testowe, które mogą być kandydatami do zestawu testów regresji i wycofać te, które już nie są odpowiednie. Testy napisane w poprzednich iteracjach, sprawdzające konkretne funkcjonalności, mogą mieć małą wartość w późniejszych iteracjach ze względu na zmiany w funkcjonalnościach lub ze względu na nowe cechy funkcjonalne zmieniające sposób zachowania się wcześniej opracowanych funkcji.

Podczas przeglądu przypadków testowych testerzy powinni rozważyć ich przydatność do automatyzacji. Zespół powinien automatyzować tyle przypadków testowych (z poprzednich i bieżącej iteracji), ile tylko jest możliwe. Pozwala to na ograniczenie ryzyka regresji przy mniejszym nakładzie pracy niż wymagałoby manualne testowanie regresji. Zmniejszony nakład pracy związany z testami regresji pozwala testerom na dokładniejsze testy nowych cech i funkcji w bieżącej iteracji.

Konieczne jest, aby testerzy mieli możliwość szybkiego identyfikowania i aktualizowania przypadków testowych z poprzednich iteracji lub wydań, na które mają wpływ zmiany wykonane w bieżącej iteracji. Podczas planowania wydania powinno zostać określone, w jaki sposób zespół projektuje, zapisuje i przechowuje przypadki testowe. Dobre praktyki projektowania testów i ich implementacji powinny być wcześnie wdrażane i konsekwentnie stosowane. Krótszy czas testowania oraz ciągłe zmiany w każdej iteracji wzmocnią wpływ złych praktyk projektowania i implementacji testów.

Wykorzystanie automatyzacji testów pozwala zespołom zwinnym na szybkie dostarczanie informacji zwrotnych na temat jakości produktu. Dobrze napisane testy automatyczne stanowią żyjącą dokumentację funkcjonalności systemu [Crispin08]. Sprawdzając testy automatyczne i ich wyniki umieszczone w systemie zarządzania konfiguracją, zgodnie z powiązaną wersją (ang. *build*) produktu, zespoły zwinne mogą przeglądać przetestowane funkcjonalności i wyniki testów dla dowolnej wersji, w dowolnym momencie.

Automatyczne testy modułowe (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 prac całego zespołu, kod nie powinien być komitowany dopóty, dopóki nie przejdą wszystkie automatyczne testy modułowe. Wyniki zautomatyzowanych testów modułowych 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 modułowe, 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 w sposób ciągły. 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 wersji w środowisku ciągłej integracji, a także za każdym razem, gdy nowa wersja jest wdrażana 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 zaktualizowane tak, by odzwierciedlać nowe kryteria 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 funkcjonalnymi.

Oprócz automatyzacji testów zautomatyzowane mogą być również następujące zadania testowe:

- generowanie danych testowych,
- wgrywanie danych testowych do systemu,
- umieszczanie wersji w środowisku testowym,
- przywrócenie środowiska testowego (np. bazy danych lub plików witryny internetowej) do konfiguracji podstawowej,
- porównywanie danych wyjściowych.

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

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 [ISTQB_FL_SYL]. Oprócz tych umiejętności tester w zespole zwinnym powinien być kompetentny w zakresie automatyzacji testów, w wytwarzaniu sterowanym testami, w wytwarzaniu sterowanym testami akceptacyjnymi, w testowaniu białoskrzynkowym i czarnoskrzynkowym oraz w testowaniu opartym na doświadczeniu.

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 posiadać dobre umiejętności interpersonalne.

Testerzy w zespołach zwinnych powinni:

- Być pozytywni (mieć pozytywne nastawienie) 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 akceptacji).
- Współpracować w zespole, pracując w parach z programistami lub innymi członkami zespołu.
- Szybko reagować na zmiany np. poprzez zmienianie, dodawanie i poprawianie przypadków testowych.
- Planować i organizować swoją własną pracę.

Stały rozwój umiejętności, w tym także umiejętności interpersonalnych, jest niezbędny dla wszystkich testerów, także dla testerów pracujących w zespołach 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 sekcjach tego sylabusa, czynności te obejmują:

- Zrozumienie, implementowanie i aktualizację strategii testów.
- Pomiar i raportowanie pokrycia testowego względem wszystkich dostępnych wymiarów pokrycia.
- Zapewnienie poprawnego wykorzystania narzędzi testowych.
- Konfigurowanie, używanie i zarządzanie środowiskiem testowym i danymi testowymi.
- Raportowanie defektów i współpracę z zespołem przy ich rozwiązywaniu.

- Szkolenie (ang. *coaching*) innych członków zespołu w różnych aspektach związanych z testowaniem.
- Zapewnienie, że odpowiednie zadania testowe są harmonogramowane podczas planowania wydań i iteracji.
- Aktywną współpracę z programistami, interesariuszami biznesowymi w celu poprawnej interpretacji wymagań, zwłaszcza w zakresie ich testowalności, spójności i kompletności.
- Aktywne uczestnictwo w retrospektywach zespołu, sugerowanie i implementowanie udoskonaleń.

W zespole zwinnym każdy członek zespołu jest odpowiedzialny za jakość produktu i odgrywa rolę w wykonywaniu zadań związanych z testami.

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

- Testerzy tak blisko pracują z deweloperami, że tracą właściwy, testerski sposób myślenia.
- 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.

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

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

Słowa kluczowe

automatyzacja wykonania testu, karta opisu testu, podejście do testów, ryzyko jakościowe, ryzyko produktowe, strategia testów, struktura do testów jednostkowych, szacowanie testów, taksonomia defektów, testowanie akceptacyjne, testowanie eksploracyjne, testowanie pielęgnowalności, testowanie regresji, 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 piramida testów.

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 pełnić rolę testera w zespole scrumowym dla danego projektu zwinnego.

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 zwinnych

FA-3.3.1 (K3) Kandydat potrafi zinterpretować odpowiednie informacje w celu wsparcia czynności testowych.

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

FA-3.3.3 (K3) Kandydat potrafi napisać przypadki testowe dla wytwarzania sterowanego testami akceptacyjnymi na podstawie danej historyjki użytkownika.

FA-3.3.4 (K3) Kandydat potrafi napisać przypadki testowe zarówno funkcjonalne, jak i niefunkcjonalne, używając technik czarnoskrzynkowych na podstawie danej historyjki użytkownika.

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 czynności w projekcie zwinnym.

3.1 Metody testowania zwinnego

Niektóre praktyki testowania można wykorzystywać w dowolnym projekcie wytwórczym (zwinnym i tradycyjnym) w celu wytworzenia produktów o wysokiej jakości. Obejmują one pisanie testów z wyprzedzeniem, aby wyrazić poprawne zachowanie, koncentrowanie się na wczesnym zapobieganiu defektom, wykrywaniu i usuwaniu usterek oraz zapewnienie, że odpowiednie typy testów będą wykonane w odpowiednim czasie na odpowiednim poziomie testów. Praktycy metodyk zwinnych dążą do wczesnego wdrożenia tych praktyk. Testerzy w projektach zwinnych pełnią kluczową rolę w kierowaniu wykorzystaniem tych praktyk testowych 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 techniki, które są wykorzystywane przez zwinne zespoły przy wykonywaniu testów na różnych poziomach testów. Każda z tych technik jest przykładem jednej z podstawowych zasad testowania mówiącej, że korzystne dla projektu jest jak najwcześniejsze testowanie i zapewnienie jakości, ponieważ w technikach tych testy są definiowane, zanim kod zostanie napisany.

Wytwarzanie sterowane testami

Wytwarzanie sterowane testami (TDD) to technika wytwarzania kodu kierowana przez zautomatyzowane przypadki testowe. Proces wytwarzania sterowanego testami jest następujący:

- Dodanie testu, który oddaje koncepcję programisty dotyczącą pożądanego działania 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 test nadal przechodzi po tym, jak kod został zrefaktoryzowany.
- Powtarzanie procesu dla następnego małego fragmentu kodu z uruchamianiem zarówno poprzednich testów, jak i nowo dodanych.

Napisane testy są przede wszystkim testami modułowymi (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, oczekiwanych wynikach. Testy te są zautomatyzowane i wykorzystywane w ciągłej integracji.

Wytwarzanie sterowane testami akceptacyjnymi

Wytwarzanie sterowane testami akceptacyjnymi [Adzic09] definiuje kryteria akceptacji oraz testy podczas tworzenia historyjek użytkownika (patrz sekcja 1.2.2). ATDD jest podejściem opartym na współpracy, które pozwala każdemu z interesariuszy zrozumieć, w jaki sposób każdy z modułów ma się zachowywać oraz czego potrzebują programiści, testerzy oraz reprezentanci biznesu, żeby zapewnić dany sposób zachowania. Proces wytwarzania sterowanego testami akceptacyjnymi został wyjaśniony w sekcji 3.3.2.

ATDD tworzy reużywalne testy do wykorzystania w testach regresji. Specjalne narzędzia wspierają tworzenie i wykonywanie takich testów, często w ramach procesu ciągłej integracji. Narzędzia te mogą łączyć się z warstwami danych i usług aplikacji, co pozwala na wykonywanie testów systemowych i akceptacyjnych. Wytwarzanie sterowane testami akceptacyjnymi pozwala na szybkie usunięcie defektów oraz na walidację zachowania cechy funkcjonalnej. Pomaga ono stwierdzić, czy kryteria akceptacji zostały spełnione dla danej funkcji.

Wytwarzanie sterowane zachowaniem (BDD)

Wytwarzanie sterowane zachowaniem [Chelimsky10] pozwala programiście na skupienie się na testowaniu kodu w oparciu o wymagane zachowanie oprogramowania. Ponieważ testy oparte są o faktyczne zachowanie oprogramowania, są one łatwiejsze do zrozumienia dla innych 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 są pewne wyniki (rezultaty).

Na podstawie tych wymagań, struktura wytwarzania sterowanego zachowaniem generuje kod, który może być wykorzystywany przez programistów do tworzenia przypadków testowych. Wytwarzanie sterowane zachowaniem pomaga programistom współpracować z innymi interesariuszami, w tym testerami, w celu zdefiniowania dokładnych testów modułowych skoncentrowanych na potrzebach biznesowych.

3.1.2 Piramida testów

System oprogramowania może być testowany na różnych poziomach testów. Typowe poziomy testów to (od najniższego do najwyższego): testy modułowe, integracji modułów, systemowe, integracji systemów i akceptacyjne (patrz [ISTQB FL_SYL], sekcja 2.2.1). Piramida testów akcentuje potrzebę posiadania dużej liczby testów na niższych poziomach (dół piramidy). Gdy wytwarzanie przechodzi do wyższych poziomów, liczba testów maleje (szczyt piramidy). Na ogół testy modułowe (jednostkowe) i integracyjne są zautomatyzowane 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 (ang. *GUI*). Koncepcja piramidy testów jest oparta 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 [Crispin08], przyporządkowują w metodykach zwinnych poziomom testów odpowiednie typy testów. Kwadranty testowe i ich modyfikacje pomagają zapewnić, że wszystkie ważne typy testów i poziomy testów są uwzględnione w cyklu życia oprogramowania. Model ten pokazuje też sposób na rozróżnienie i opisanie typów testów wszystkim interesariuszom, włącznie z programistami, testerami i przedstawicielami biznesu.

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

- Kwadrant Q1 to poziom modułowy (jednostkowy), który jest zorientowany na technologię i wspomaga deweloperów. W tym kwadrancie znajdują się testy modułowe. Testy te powinny być zautomatyzowane i włączone do procesu ciągłej integracji.
- Kwadrant Q2 to poziom systemowy, który jest zorientowany na biznes i potwierdza zachowanie produktu. W tym kwadrancie zawarte są testy funkcjonalne, przykłady, testowanie historyjek, prototypy oparte na doświadczeniu użytkowników oraz symulacje. Te testy sprawdzają kryteria akceptacji. Mogą one być manualne lub zautomatyzowane. Często są tworzone podczas opracowywania historyjek użytkownika i tym samym poprawiają jakość historyjek. Są użyteczne do tworzenia zestawów automatycznych testów regresji.
- Kwadrant Q3 to poziom systemowy lub akceptacji użytkownika, który jest zorientowany na biznes i zawiera testy, które krytykują produkt używając realistycznych scenariuszy i danych. Ten kwadrant obejmuje testy eksploracyjne, scenariusze, przepływy procesów, testowanie użyteczności, testy akceptacyjne użytkownika, testy alfa i beta. Te testy są często manualne i zorientowane na użytkownika.
- Kwadrant Q4 to poziom systemowy lub akceptacji operacyjnej, który jest zorientowany na technologię i zawiera testy krytykujące produkt. Ten kwadrant obejmuje testy wydajnościowe, obciążeniowe, przeciążające, testowanie skalowalności, zabezpieczeń, pielęgnowalności, zarządzania pamięcią, testowanie kompatybilności i współdziałania, migracji danych, infrastruktury oraz testowanie odzyskiwania. Testy te są często automatyzowane.

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

3.1.4 Rola testera

W niniejszym sylabusie 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 [Aalst13].

Praca zespołowa

Praca zespołowa to podstawowa zasada w wytwarzaniu zwinnym. Zwinność kładzie nacisk na podejście „cały zespół" składający się z programistów, testerów i przedstawicieli biznesu, pracujących razem. Poniżej podano najlepsze praktyki organizacyjne i behawioralne w zespołach scrumowych:

- Interdyscyplinarność: każdy członek zespołu wnosi do niego inny zestaw umiejętności. Zespół pracuje wspólnie nad strategią testów, specyfikacją testów, wykonaniem testów i raportowaniem wyników testów.
- Samoorganizacja zespołu: zespół może składać się z samych deweloperów, ale – jak podano w sekcji 2.1.5 – najlepiej byłoby, gdyby do zespołu dołączył jeden lub więcej testerów.
- Wspólna lokalizacja: testerzy pracują w jednym miejscu (w środowisku fizycznym lub wirtualnym) razem z deweloperami i właścicielem produktu.
- Współpraca: testerzy współpracują z członkami swojego zespołu, innymi zespołami, interesariuszami, właścicielem produktu i Scrum Masterem.
- Umocowanie (upoważnienie): decyzje techniczne dotyczące projektowania i testowania są podejmowane przez członków zespołu jako całość (programiści, testerzy i Scrum Master), we współpracy z właścicielem produktu i w razie potrzeby z innymi zespołami.
- Zaangażowanie: tester jest zobowiązany do kwestionowania i oceniania zachowania oraz charakterystyk produktu, z uwzględnieniem oczekiwań i potrzeb klientów oraz użytkowników.
- Transparentność: postęp prac programistycznych i testerskich jest widoczny na zwinnej tablicy zadań (patrz sekcja 2.2.1).
- Wiarygodność: tester musi zapewnić wiarygodność strategii testów, 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 podstawie sukcesów i porażek.
- Elastyczny: testowanie musi umożliwiać reagowanie na zmiany, podobnie jak inne czynności w projektach zwinnych.

Te najlepsze praktyki maksymalizują prawdopodobieństwo sukcesu testowania w projektach scrumowych.

Sprint Zero

Sprint zero to pierwsza iteracja projektu, podczas której ma miejsce wiele czynności wstępnych (patrz sekcja 1.2.5). Podczas tej iteracji tester współpracuje z zespołem

wykonując następujące czynności:

- Określenie zakresu projektu (tzn. backlogu produktu).
- Stworzenie początkowej architektury systemu i prototypów wysokiego poziomu.
- Zaplanowanie, zakup i instalacja niezbędnych narzędzi (np. do zarządzania testami, zarządzania defektami, automatyzacji testów i do ciągłej integracji).
- Stworzenie wstępnej strategii testów dla wszystkich poziomów testów uwzględniającej (między innymi) takie zagadnienia jak: zakres testów, ryzyko techniczne, typy testów (patrz sekcja 3.1.3) oraz cele dotyczące pokrycia.
- Wykonanie wstępnej analizy ryzyk jakościowych (patrz sekcja 3.2.1).
- Zdefiniowanie metryk testowych do pomiaru procesu testowego, postępu testów w projekcie i jakości produktu.
- Określenie definicji ukończenia.
- Utworzenie tablicy zadań (patrz sekcja 2.2.1).
- Określenie, kiedy należy kontynuować lub zakończyć testy przed dostarczeniem systemu do klienta.

Sprint zero nadaje kierunek testom: co testowanie powinno osiągnąć i jak ma to osiągnąć podczas sprintów.

Integracja

W projektach zwinnych celem jest dostarczanie wartości dla klienta w sposób ciągły (najlepiej w każdym sprincie). Aby to umożliwić, strategia integracji powinna uwzględniać zarówno projektowanie, jak i testowanie. Aby umożliwić strategię ciągłego testowania dla dostarczanej funkcjonalności i cech, 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 omówione w sekcji 1.2.5.

Efektem planowania sprintu jest zbiór zadań do umieszczenia na tablicy zadań. Wykonanie każdego z zadań powinno zabierać jeden lub dwa dni pracy. Ponadto należy śledzić wszystkie problemy związane z testowaniem, aby utrzymać stały przepływ testów.

Praktyki testowania zwinnego

Istnieje wiele praktyk, które mogą być użyteczne dla testerów w zespołach scrumowych. Niektóre z nich obejmują:

- Pracę w parach: dwóch członków zespołu (np. tester i programista, dwóch testerów lub tester i właściciel produktu) siedzi razem przy jednym stanowisku pracy, aby wykonać testy lub wykonać inne zadanie dla sprintu.
- Przyrostowe projektowanie testów: przypadki testowe i karty opisu testów są

stopniowo budowane na podstawie historyjek użytkownika i innych podstaw testów, zaczynając od prostych testów i przechodząc do bardziej złożonych.
- Mapy myśli: mapy myśli są użytecznym narzędziem podczas testowania [Crispin08]. Na przykład testerzy mogą używać mapowania myśli w celu określenia, które sesje testowe należy wykonać, aby pokazać strategie testów i opisać dane testowe.

Praktyki te stanowią dodatek do innych praktyk omówionych w niniejszym sylabusie oraz w rozdziale 4 Sylabusa Poziomu Podstawowego [ISTQB_FL_SYL].

3.2 Ocena ryzyk jakościowych produktu i szacowanie pracochłonności

Typowym celem testowania we wszystkich projektach (zarówno zwinnych, jak i tradycyjnych) jest redukcja ryzyka występowania problemów związanych z jakością produktu do akceptowalnego poziomu przed jego wydaniem. Testerzy w projektach zwinnych mogą wykorzystywać te same rodzaje technik, które są stosowane w tradycyjnych projektach do identyfikowania ryzyk jakościowych (lub ryzyka produktowego) , szacowania związanego z nim poziomu ryzyka, oszacowywania nakładu pracy potrzebnego do dostatecznej redukcji tego ryzyka, a następnie złagodzenie tego ryzyka poprzez zaprojektowanie, implementację i wykonanie testów. Biorąc jednak pod uwagę krótkie iteracje i dużą ilość zmian w projektach zwinnych, niezbędne są pewne adaptacje tych technik.

3.2.1 Szacowanie ryzyk jakościowych w projekcie zwinnym

Jednym z wielu wyzwań związanych z testowaniem jest właściwy dobór, alokacja i priorytetyzacja warunków testowych. Obejmuje to określenie odpowiedniego nakładu pracy, jaki należy przeznaczyć na pokrycie testami każdego warunku, a także uporządkowanie uzyskanych przypadków testowych tak, by zoptymalizować efektywność i wydajność pracy związanej z testowaniem, którą trzeba wykonać. Identyfikacja, analiza ryzyka oraz strategie łagodzenia ryzyka mogą być wykorzystywane przez testerów w zespołach zwinnych do oszacowania akceptowalnej liczby przypadków testowych do wykonania. Tym niemniej wiele oddziałujących na siebie ograniczeń i zmiennych może wymagać kompromisów.

Ryzyko to możliwość wystąpienia negatywnego lub niepożądanego rezultatu lub zdarzenia. Poziom ryzyka jest określany poprzez ocenę prawdopodobieństwa wystąpienia ryzyka oraz jego wpływu. Jeżeli podstawowym skutkiem potencjalnego problemu jest spadek jakości produktu, wówczas potencjalne problemy są określane ryzykami jakościowymi lub produktowymi. Jeżeli podstawowym skutkiem potencjalnego problemu jest zagrożenie powodzenia projektu, wówczas potencjalne problemy są określane ryzykami projektowymi lub ryzykami planowania [Black07] [vanVeenendaal12].

W projektach zwinnych analiza ryzyka jakościowego (produktowego) występuje w dwóch miejscach.

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

Przykłady ryzyk jakościowych dla systemu obejmują:

- Nieprawidłowe obliczenia w raportach (ryzyko funkcjonalne związane z dokładnością).
- Wolna reakcja na dane wprowadzane przez użytkownika (ryzyko niefunkcjonalne związane z wydajnością i czasem odpowiedzi).
- Trudności w zrozumieniu ekranów i pól (ryzyko niefunkcjonalne związane z użytecznością i zrozumiałością).

Jak wspomniano powyżej, iterację rozpoczyna planowanie iteracji, którego rezultatem są oszacowane zadania na tablicy zadań. Zadania te mogą zostać spriorytetyzowane częściowo według związanego z nimi poziomu ryzyka jakościowego. Zadania posiadające wyższy poziom ryzyka powinny rozpoczynać się wcześniej i wymagają większego wysiłku testowego. Zadania związane z niższym poziomem ryzyka powinny rozpoczynać się później i wymagać mniejszego nakładu pracy związanej z testowaniem.

Przykład tego, w jaki sposób można przeprowadzić proces analizy ryzyka jakościowego w projekcie zwinnym podczas planowania iteracji, został opisany w poniższych krokach:

1. Zebranie członków zespołu zwinnego, razem z testerem (testerami).
2. Sporządzenie listy wszystkich zaległych pozycji z backlogu produktu dla bieżącej iteracji (np. na tablicy zadań).
3. Zidentyfikowanie ryzyka jakościowego związanego z każdą pozycją, biorąc pod uwagę wszystkie istotne charakterystyki jakościowe.
4. Ocena każdego zidentyfikowanego ryzyka, która obejmuje dwie czynności: kategoryzację ryzyka oraz określenie poziomu ryzyka na podstawie wpływu i prawdopodobieństwa wystąpienia usterek.
5. Ustalenie zakresu testów proporcjonalnie do poziomu ryzyka.
6. Wybór odpowiedniej techniki (technik) testowania w celu łagodzenia każdego z ryzyk, w oparciu o ryzyko, poziom ryzyka oraz odpowiadającą mu charakterystykę jakościową.

Następnie testerzy projektują, implementują oraz wykonują testy, żeby złagodzić zidentyfikowane ryzyka. Obejmuje to całość cech (funkcji), zachowań, charakterystyk jakościowych oraz tych wszystkich atrybutów, które wpływają na satysfakcję klientów, użytkowników i interesariuszy.

Przez cały czas trwania projektu zespół powinien być świadomy, że dodatkowe informacje mogą zmienić zbiór ryzyk i/lub poziom ryzyk związanych ze znanymi ryzykami jakościowymi. Ponadto zaleca się przeprowadzanie okresowej korekty analizy

ryzyk jakościowych produktu, a co za tym idzie wprowadzenie odpowiednich zmian w testach. Poprawki zawierają nowe ryzyka, ponowną ocenę poziomów istniejących ryzyk oraz ocenę skuteczności działań łagodzących ryzyka.

Ryzyko jakościowe może być także łagodzone, zanim 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 Szacowanie nakładu pracy testowej w oparciu o treść i ryzyko

Podczas planowania wydania zespół zwinny szacuje pracochłonność wymaganą do ukończenia wydania. Szacunki te dotyczą również pracochłonności testowania. Często wykorzystywaną przez zespoły zwinne techniką jest poker planistyczny, technika bazująca na konsensusie. Właściciel produktu lub klient czyta historyjkę użytkownika osobom estymującym. Każdy z estymujących ma talię kart z wartościami identycznymi z pierwszymi wartościami ciągu Fibonacciego (tj. 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …) lub z wartościami innego ciągu rosnącego (np. rozmiary koszulek 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 funkcji i w ramach potrzeb zadają pytania właścicielowi produktu. Aspekty takie jak wysiłek włożony w wytwarzanie i testowanie, złożoność historyjki i zakres testowania odgrywają rolę w szacowaniu. W związku z tym zaleca się uwzględnienie poziomu ryzyka elementu backlogu, oprócz priorytetu określonego przez właściciela produktu, przed rozpoczęciem sesji pokera planistycznego. Gdy funkcja została w pełni omówiona, każdy estymujący wybiera jedną kartę, która reprezentuje jego oszacowanie. Wszystkie karty są następnie ujawniane w tym samym czasie. Jeśli wszyscy estymujący wybrali tę samą wartość, staje się ona wartością szacunkową. Jeśli nie, estymujący omawiają różnice w szacunkach, po czym runda pokera jest powtarzana aż do osiągnięcia porozumienia, albo w drodze konsensusu, albo poprzez zastosowanie zasad (użycie mediany, użycie najwyższego wyniku), aby ograniczyć liczbę rund pokera. Dyskusje te zapewniają wiarygodne oszacowanie wysiłku potrzebnego do ukończenia pozycji backlogu produktu wymaganych przez właściciela produktu i pomagają poprawić zbiorową wiedzę na temat tego, co należy zrobić [Cohn04].

3.3 Techniki w projektach zwinnych

Wiele technik testowych i poziomów testów, które stosuje się w tradycyjnych projektach, można również zastosować w projektach zwinnych. W przypadku projektów zwinnych istnieją jednak pewne szczególne względy i różnice w technikach testowania, terminologii i dokumentacji, które należy wziąć pod uwagę.

3.3.1 Kryteria akceptacji, odpowiednie pokrycie i inne informacje dotyczące testowania

Projekty zwinne określają początkowe wymagania w postaci historyjek użytkownika, w uporządkowanym pod względem ważności backlogu na początku projektu. Początkowo wymagania są krótkie i zgodne z predefiniowanym formatem (patrz sekcja 1.2.2). Wymagania niefunkcjonalne, takie jak wydajność i użyteczność, są równie istotne i mogą być określone jako osobne historyjki użytkownika lub połączone z innymi funkcjonalnymi historyjkami użytkownika. Wymagania niefunkcjonalne mogą być zgodne z predefiniowanym formatem lub standardem, takim jak [ISO25000] lub ze standardem branżowym.

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

- doświadczenie z poprzednich projektów,
- istniejące funkcje, cechy i charakterystyki jakościowe systemu,
- kod, architektura oraz projekt,
- profile użytkowników (kontekst, konfiguracja systemu i zachowanie użytkownika),
- informacja o defektach z obecnych i poprzednich projektów,
- kategoryzacja defektów w taksonomii defektów,
- dostępne standardy (np. [DO-178C] dla oprogramowania awioniki),
- ryzyka jakościowe (patrz sekcja 3.2.1).

Podczas każdej iteracji programiści tworzą kod, który implementuje, z uwzględnieniem odpowiednich cech jakościowych, funkcje oraz cechy opisane w historyjkach użytkownika. Kod ten jest weryfikowany i walidowany za pomocą testów akceptacyjnych. Aby kryteria akceptacji były testowalne, powinny one dotyczyć następnujących zagadnień (o ile są one istotne) [Wiegers13]:

- Zachowanie funkcjonalne: zewnętrznie obserwowalne zachowanie z działaniami użytkownika jako danymi wejściowymi operującymi w określonych konfiguracjach.
- Charakterystyki jakościowe: sposób, w jaki system wykonuje określone zachowanie. Charakterystyki jakościowe można również nazywać atrybutami jakościowymi lub wymaganiami niefunkcjonalnymi. Typowe charakterystyki jakościowe to wydajność, niezawodność, użyteczność itp.
- Scenariusze (przypadki użycia): sekwencja czynności pomiędzy zewnętrznym aktorem (często użytkownikiem) a systemem, w celu osiągnięcia określonego celu lub zadania biznesowego.
- Reguły biznesowe: czynności, które mogą być wykonywane w systemie tylko w określonych warunkach zdefiniowanych przez zewnętrzne procedury i ograniczenia (np. procedury stosowane przez firmy ubezpieczeniowe do obsługi roszczeń ubezpieczeniowych).
- Interfejsy zewnętrzne: opisy połączeń pomiędzy rozwijanym systemem a światem zewnętrznym. Interfejsy zewnętrzne można podzielić na różne typy

(interfejs użytkownika, interfejs do innych systemów itp.).
- Ograniczenia: wszelkie ograniczenia projektowe i implementacyjne, które ograniczają możliwości programisty. Urządzenia z oprogramowaniem wbudowanym często muszą spełniać wymagania fizyczne takie jak rozmiar, waga i 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).

Oprócz historyjek użytkownika i powiązanych z nimi kryteriów akceptacji, dla testera istotne są również inne informacje, w tym:
- Jak system ma działać i jak będzie używany?
- Interfejsy systemowe, które mogą być używane/dostępne do testowania systemu.
- Czy obecne wsparcie narzędziowe jest wystarczające?
- Czy tester posiada wystarczającą wiedzę i umiejętności do przeprowadzenia niezbędnych testów?

Testerzy często odkrywają potrzebę uzyskania dodatkowych informacji (np. pokrycie kodu) w trakcie trwania iteracji i powinni współpracować z pozostałymi członkami zespołu zwinnego w celu uzyskania tych informacji. Posiadanie odpowiednich informacji odgrywa dużą rolę w określeniu, czy poszczególne zadania mogą zostać uznane za zakończone. Ta koncepcja definicji ukończenia jest krytyczna dla projektów zwinnych i można ją zastosować na wiele różnych sposobów, tak jak zostało to omówione w poniższych sekcjach.

Poziomy testów

Każdy poziom testów ma swoją własną definicję ukończenia. Poniższa lista przedstawia przykłady tego kryterium dla różnych poziomów testów.
- Testy modułowe
  - 100% pokrycia decyzji tam, gdzie jest to możliwe, z dokładnym przeglądem każdej niewykonalnej ścieżki.
  - Analiza statyczna wykonana dla całego kodu.
  - Brak nierozwiązanych głównych defektów (uporządkowanych zgodnie z priorytetami i wagami).
  - Brak znanego, nieakceptowalnego długu technicznego pozostałego w projekcie i kodzie [Jones11].
  - Przegląd całego kodu, testów modułowych i ich wyników.
  - Wszystkie testy modułowe zautomatyzowane.
  - Ważne charakterystyki jakościowe mieszczą się w wyznaczonych granicach (np. wydajność).
- Testy integracyjne
  - Przetestowane wszystkie wymagania funkcjonalne, w tym zarówno testy pozytywne, jak i negatywne, z pewną liczbą testów opartych na wielkości, złożoności i ryzykach.
  - Przetestowane wszystkie interfejsy pomiędzy modułami.

  - Pokryte wszystkie ryzyka jakościowe zgodnie z uzgodnionym zakresem testów.
  - Brak nierozwiązanych poważnych usterek (uporządkowanych według ryzyka i ważności).
  - Zaraportowane wszystkie znalezione defekty.
  - Wszystkie testy regresji są zautomatyzowane, o ile to możliwe, a wszystkie zautomatyzowane testy są przechowywane we wspólnym repozytorium.
- Testy systemowe
  - Kompleksowe (ang. *end-to-end*) testy historyjek użytkownika, cech i funkcji.
  - Uwzględniono wszystkie grupy użytkowników.
  - Pokryte najważniejsze charakterystyki jakościowe systemu (np. wydajność, odporność, niezawodność).
  - Testowanie przeprowadzone w środowisku (środowiskach) podobnym do produkcyjnego, obejmujące cały sprzęt i oprogramowanie dla wszystkich obsługiwanych konfiguracji, w możliwym zakresie.
  - Wszystkie ryzyka jakościowe pokryte zgodnie z uzgodnionym zakresem testów.
  - Zautomatyzowane wszystkie testy regresji (tam, gdzie tylko jest to możliwe), a wszystkie testy automatyczne przechowywane są we wspólnym repozytorium.
  - Wszystkie znalezione defekty zostały zaraportowane i - jeżeli to możliwe - naprawione.
  - Brak nierozwiązanych poważnych usterek (uporządkowanych według ryzyka i ważności).

Historyjka użytkownika

Definicja ukończenia dla historyjek użytkownika może być określona przez następujące kryteria:

- Historyjki użytkownika wybrane dla iteracji są kompletne, zrozumiałe dla zespołu i mają szczegółowe, testowalne kryteria akceptacji.
- Wszystkie elementy historyjki użytkownika zostały określone i zweryfikowane, włącznie z testami akceptacyjnymi dla historyjki.
- Zadania niezbędne do wdrożenia i przetestowania wybranych historyjek użytkownika zostały zidentyfikowane i oszacowane przez zespół.

Cecha funkcjonalna

Definicja ukończenia dla wykonanych cech funkcjonalnych, które mogą obejmować wiele historyjek użytkownika lub opowieści, może obejmować:

- Wszystkie składające się na cechę funkcjonalną historyjki użytkownika, w tym ich kryteria akceptacji, są zdefiniowane i zatwierdzone przez klienta.
- Projekt jest kompletny, bez znanego długu technicznego.
- Kod jest kompletny, bez znanego długu technicznego lub niedokończonej refaktoryzacji.

- Testy modułowe zostały wykonane i osiągnęły określony poziom pokrycia.
- Testy integracyjne i testy systemowe dla danej cechy funkcjonalnej zostały wykonane zgodnie ze zdefiniowanymi kryteriami pokrycia.
- Wszystkie ważne usterki zostały naprawione.
- Dokumentacja danej cechy funkcjonalnej jest kompletna, co może obejmować informację o wydaniu, podręczniki użytkownika i funkcje pomocy online.

Iteracja

Definicja ukończenia dla iteracji może obejmować następujące elementy:

- Wszystkie cechy (funkcje) dla iteracji są gotowe i indywidualnie przetestowane zgodnie z kryteriami testowania tych cech.
- Wszystkie niekrytyczne defekty, które nie mogą zostać naprawione ze względu na ograniczenia iteracji, zostały dodane do backlogu produktu i spriorytetyzowane.
- Integracja wszystkich cech (funkcji) dla danej iteracji została wykonana i przetestowana.
- Dokumentacja została sporządzona, sprawdzona i zatwierdzona.

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

Wydanie

Definicja ukończenia dla wydania, które może obejmować wiele iteracji, może uwzględniać następujące obszary:

- Pokrycie: wszystkie istotne elementy podstawy testów dla całej zawartości wydania zostały pokryte testami. Adekwatność pokrycia zależy od tego, co jest nowe lub zmienione, jego złożoności i rozmiaru zmienionych elementów oraz powiązane z tym ryzyko awarii.
- Jakość: natężenie defektów (tzn. liczba wykrywanych defektów na dzień lub na transakcję), gęstość defektów (tzn. liczba znalezionych defektów w porównaniu do liczby historyjek użytkownika, pracochłonności i/lub atrybutów jakości), szacowana liczba pozostałych defektów mieszczą się w akceptowalnych granicach. Konsekwencje nierozwiązanych i pozostawionych defektów (np. waga i priorytet) są zrozumiałe i akceptowalne. Poziom ryzyka rezydualnego związanego z każdym zidentyfikowanym ryzykiem jakościowym jest zrozumiały i akceptowalny.
- Czas: jeżeli wcześniej określony termin dostawy został osiągnięty, biznesowe czynniki wydania lub niewydania powinny zostać rozważone.
- Koszt: szacowany koszt cyklu życia powinien zostać wykorzystany do obliczenia zwrotu z inwestycji dla dostarczonego systemu (tzn. obliczony koszt wytworzenia i pielęgnacji powinien być znacząco niższy niż oczekiwany sumaryczny przychód ze sprzedaży produktu). Ze względu na liczbę defektów przepuszczonych na produkcję większa część kosztów w cyklu życia zwykle przypada na koszt pielęgnacji po wdrożeniu.

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 wchodzą: programista, tester i przedstawiciel biznesu [Adzic09] i mogą być wykonywane ręcznie lub zautomatyzowane.

Pierwszym krokiem jest warsztat, podczas którego historyjka użytkownika jest analizowana, omawiana i opisywana przez programistów, testerów i przedstawicieli jednostek biznesowych. W tym procesie naprawiane są wszelkie braki, niejednoznaczności oraz błędy.

Następnym krokiem jest zaprojektowanie testów. Może to zostać wykonane wspólnie przez zespół lub indywidualnie przez testera. W każdym przypadku niezależna osoba, taka 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 zamiennie. Praca rozpoczyna się od podstawowych przykładów i pytań otwartych.

Zazwyczaj pierwsze przypadki testowe są pozytywne. Potwierdzają one prawidłowe zachowanie (bez wyjątków lub obsługi błędów), obejmujące sekwencję czynności wykonywanych, jeśli wszystko przebiega zgodnie z oczekiwaniami. Po wykonaniu testów pozytywnych zespół powinien zaprojektować testy negatywne, obejmujące również atrybuty niefunkcjonalne (np. wydajność, użyteczność). Testy pisane są w sposób zrozumiały dla interesariusza. Zawierają one zdania budowane w języku naturalnym, obejmują niezbędne warunki wstępne (jeśli takie występują), dane wejściowe oraz powiązane dane wyjściowe.

Przykłady muszą pokrywać wszystkie cechy historyjki użytkownika i nie powinny jej uzupełniać. Oznacza to, że nie powinien istnieć przykład opisujący aspekt historyjki użytkownika, który nie został udokumentowany w samej historyjce. 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 przez testerów równolegle z czynnościami programistycznymi. Tak jak programiści programują w oparciu o historyjki użytkownika i kryteria akceptacji, tak samo testerzy projektują testy również w oparciu o historyjki użytkownika i ich kryteria akceptacji. Niektóre testy, takie jak testy eksploracyjne oraz inne testy oparte na doświadczeniu, są tworzone później, podczas fazy wykonywania testów (patrz sekcja 3.3.4). Do projektowania testów testerzy mogą wykorzystywać tradycyjne techniki projektowania testów czarnoskrzynkowych, takie jak klasy równoważności, analiza wartości brzegowych,

tablice decyzyjne oraz testowanie przejść pomiędzy stanami. Na przykład analiza wartości brzegowych może zostać wykorzystana do wyboru wartości testowych, gdy klient jest ograniczony liczbą produktów, które może wybrać podczas zakupu.

W wielu sytuacjach wymagania niefunkcjonalne mogą być dokumentowane jako historyjki użytkownika. Czarnoskrzynkowe techniki projektowania testów (np. analiza wartości brzegowych) mogą zostać wykorzystane do tworzenia testów niefunkcjonalnych charakterystyk jakościowych. Historyjka użytkownika może zawierać wymagania dotyczące wydajności lub niezawodności. Na przykład dane wykonanie nie może przekroczyć limitu czasu lub pewna liczba operacji może zakończyć się niepowodzeniem mniej niż określoną liczbę razy.

Więcej informacji na temat wykorzystania technik czarnoskrzynkowych znajduje się w Sylabusie Poziomu Podstawowego [ISTQB_FL_SYL] oraz w Sylabusie Poziomu Zaawansowanego - Analityk Testów [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 ograniczoną szczegółowość historyjek użytkownika. W celu osiągnięcia najlepszych rezultatów, testowanie eksploracyjne powinno być połączone z innymi technikami opartymi na doświadczeniu jako części reaktywnej strategii testów, połączonej z innymi strategiami testów, takimi jak strategia analityczna oparta na ryzyku, strategia analityczna oparta na wymaganiach, testowanie oparte na modelu czy testowanie regresji. Strategie testowania oraz ich łączenie omówiono w [ISTQB_FL_SYL].

W testowaniu eksploracyjnym projektowanie i wykonywanie testów wykonuje się równocześnie, w oparciu o wcześniej przygotowaną kartę opisu testu. Karta opisu testu określa warunki testowe, które powinny zostać pokryte podczas ograniczonej czasowo sesji testowej. Podczas testów eksploracyjnych wyniki ostatnich testów wyznaczą kierunek następnych testów. Do projektowania testów można wykorzystać te same techniki biało- i czarnoskrzynkowe, co w przypadku wstępnie zaprojektowanych testów.

Karta opisu testu 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, aby rozpocząć wykonywanie testu.
- Priorytet: względna istotność tej karty, oparta na priorytecie powiązanej historyjki użytkownika lub poziomie ryzyka.
- Odniesienia: specyfikacja (np. historyjka użytkownika), ryzyko i inne źródła informacji.
- Dane: wszelkie dane potrzebne do realizacji karty.
- Czynności: lista pomysłów na to, co aktor może chcieć wykonywać w systemie (np. „Zaloguj się do systemu jako superużytkownik”), a także na to, co mogłoby

być interesujące do przetestowania (zarówno w testach pozytywnych, jak i negatywnych).
- Uwagi wyroczni testowej: jak ocenić produkt, aby określić poprawne wyniki (np. uchwycić to, co dzieje się na ekranie i porównać to z tym, co napisano
w podręczniku użytkownika).
- Warianty: alternatywne działania i oceny uzupełniające pomysły opisane w ramach działań.

Do zarządzania testami eksploracyjnymi wykorzystuje się metodę zwaną zarządzaniem testami w sesjach. Sesja to nieprzerwany 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 charakterystyki),
- głębokie pokrycie (przypadki skrajne, scenariusze, interakcje).

Jakość testów zależy od zdolności testerów do zadawania właściwych pytań dotyczących tego, co należy przetestować. Przykłady obejmują następujące kwestie:
- Co jest najważniejsze do odkrycia w systemie?
- W jaki sposób system może ulec awarii?
- Co się stanie, gdy …?
- Co powinno się stać, gdy …?
- Czy spełnione są potrzeby, wymagania i oczekiwania klienta?
- Czy można system zainstalować (i usunąć, gdy jest to konieczne) we wszystkich obsługiwanych ścieżkach aktualizacji?

Podczas wykonywania testów tester wykorzystuje swoją kreatywność, intuicję, wiedzę oraz umiejętności do wyszukiwania potencjalnych problemów w produkcie. Tester powinien także posiadać szeroką wiedzę na temat systemu, dobrze rozumieć testowane oprogramowanie, dziedzinę biznesową, sposób korzystania z oprogramowania oraz tego, 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 ocenić ich wyniki [Hendrickson]. Przykłady heurystyk to:
- granice,
- CRUD (*Create, Read, Update, Delete* - utwórz, odczytaj, aktualizuj i usuń),
- warianty konfiguracji,
- przerwy (np. wylogowanie, wyłączenie lub ponowne uruchomienie).

Ważne jest, aby tester dokumentował proces testów tak dokładnie, jak jest to możliwe. W przeciwnym razie powrót i odkrycie sposobu wykrycia problemu w systemie może być znacząco utrudnione.
Poniższa lista zawiera przykłady informacji, które mogą być przydatne do udokumentowania:
- Pokrycie testami: jakich danych wejściowych użyto, ile z nich zostało objętych testami i ile pozostało do przetestowania.

- Notatki z oceny: obserwacje podczas testów, czy testowany system i testowana funkcja są stabilne, czy znaleziono jakieś defekty, jaki jest planowany następny krok zgodnie z bieżącymi obserwacjami i lista pomysłów.
- Lista ryzyk/strategii: które z najważniejszych ryzyk zostały pokryte, a które pozostały; czy kontynuuje się początkową strategię, czy wymaga ona jakichś zmian.
- Problemy, zapytania i anomalie: wszystkie nieoczekiwane zachowania, pojawiające się pytania dotyczące efektywności podejścia, wszelkie obawy dotyczące pomysłów/podejścia do testów, środowiska testowego, danych testowych, niezrozumienie funkcji, skryptu testowego lub testowanego systemu.
- Rzeczywiste zachowanie: zapisy rzeczywistego zachowania systemu, które należy zachować (np. wideo, zrzuty ekranu, pliki danych wyjściowych).

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) w taki sposób, aby interesariusze łatwo mogli zrozumieć bieżący status wszystkich przeprowadzonych testów.

3.4 Narzędzia w projektach zwinnych

Narzędzia opisane w [ISTQB-FL_SYL] są wciąż odpowiednie i używane przez testerów w zespołach zwinnych. Nie wszystkie z nich są wykorzystywane w ten sam sposób, a niektóre z nich mają większe znaczenie w projektach zwinnych niż w projektach tradycyjnych. Na przykład narzędzia do zarządzania testami, narzędzia do zarządzania wymaganiami czy narzędzia do zarządzania incydentami (narzędzia do śledzenia defektów) mogą być używane przez zespoły zwinne. Niektóre zespoły zwinne decydują się na wszechstronne narzędzie (np. narzędzie do zarządzania cyklem życia aplikacji lub zarządzania zadaniami), które zapewnia funkcje istotne dla wytwarzania zwinnego, takie jak tablice zadań, wykresy spalania czy historyjki użytkownika. Narzędzia do zarządzania konfiguracją są ważne dla testerów w zespołach zwinnych ze względu na dużą liczbę testów automatycznych na wszystkich poziomach, a także potrzebę przechowywania i zarządzania powiązanymi z nimi artefaktami testów automatycznych.

Oprócz narzędzi opisanych w Sylabusie Poziomu Podstawowego [ISTQB-FL_SYL] testerzy w projektach zwinnych używają też narzędzi opisanych w poniższych sekcjach. Narzędzia te są używane przez cały zespół w celu zapewnienia współpracy w zespole i wymiany informacji, które są kluczowe dla podejścia zwinnego.

3.4.1 Narzędzia do zarządzania zadaniami i śledzenia

W niektórych przypadkach zespoły zwinne używają fizycznych tablic historyjek/zadań (np. białych lub korkowych) 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 następującym celom:

- Rejestrowanie historyjek i związanych z nimi zadań wytwórczych i testowych, by mieć pewność, że nic nie zostanie utracone 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 zapewnić pełny obraz nakładu pracy zespołu niezbędnego do jej implementacji.
- Agregowanie aktualizacji statusu zadań dla programistów i testerów w miarę wykonywania przez nich pracy, i jednoczesne dostarczanie aktualnego statusu każdej historyjki, iteracji i całego wydania.
- Dostarczanie wizualnej reprezentacji (z użyciem metryk, wykresów, tablic wskaźników) aktualnego stanu każdej historyjki użytkownika, iteracji oraz wydania, umożliwiając wszystkim interesariuszom, w tym osobom w zespołach rozproszonych, szybkie sprawdzenie statusu danej historyjki.
- Integracja z narzędziami do zarządzania konfiguracją, które mogą umożliwić automatyczne komitowanie wgranego na serwer kodu (ang. *commit*) i wersji w odniesieniu do zadań, a także - w pewnych sytuacjach - automatyczne aktualizacje statusu zadań.

3.4.2 Narzędzia do komunikacji i wymiany informacji

Oprócz poczty elektronicznej, zespoły zwinne często korzystają z trzech dodatkowych rodzajów narzędzi wspierających komunikację i dzielenie się informacjami: wiki, komunikatory internetowe i współdzielenie pulpitu.

Wiki umożliwia zespołom tworzenie i współdzielenie internetowej bazy wiedzy na temach różnych aspektów projektu, zawierającej m.in.:

- Diagramy cech produktu, dyskusje na temat cech, diagramy prototypów, zdjęcia tablic z zapisami dyskusji i inne informacje.
- Informacje o narzędziach i/lub technikach wytwarzania i testowania, które zostały uznane za użyteczne przez innych członków zespołu.
- Metryki, wykresy i tablice wskaźników 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 w przypadku komunikatorów internetowych oraz poczty e-mail, ale w sposób, który jest udostępniany wszystkim innym członkom zespołu.

Komunikatory internetowe, narzędzia do telekonferencji i czaty wideo dostarczają następujące korzyści:

- Umożliwiają bezpośrednią komunikację w czasie rzeczywistym pomiędzy członkami zespołu, co jest ważne zwłaszcza w zespołach rozproszonych.
- Angażują zespoły rozproszone podczas codziennych spotkań na stojąco (ang. *standup meetings*).
- Zmniejszają kwoty rachunków telefonicznych dzięki wykorzystaniu technologii

VOIP, niwelując problem ograniczania kosztów, co mogłoby wpłynąć na ograniczenie komunikacji członków rozproszonych zespołów.

Narzędzia do współdzielenia pulpitu i przechwytywania dostarczają następujące 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 w celu uzupełnienia i rozszerzenia, ale nie zastąpienia bezpośredniej komunikacji w zespołach zwinnych.

3.4.3 Narzędzia do budowy i dystrybucji oprogramowania

Jak omówiono powyżej 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 wersji. Zastosowania, 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 wprowadzenie zmian bez utraty informacji historycznych. Główne typy systemów kontroli wersji obejmują scentralizowane systemy kontroli źródła i rozproszone systemy kontroli wersji. Wielkość zespołu, jego struktura, lokalizacja i wymagania dotyczące integracji z innymi narzędziami określają, który system kontroli wersji jest odpowiedni dla konkretnego projektu zwinnego.

3.4.5 Narzędzia do projektowania, implementacji i wykonywania testów

Niektóre narzędzia są użyteczne dla testera zwinnego w określonych momentach procesu testowania oprogramowania. Jakkolwiek większość z tych narzędzi nie jest nowa ani specyficzna dla procesu zwinnego, to jednak dostarczają one istotnych możliwości biorąc pod uwagę szybkie i częste zmiany w projektach zwinnych.

- Narzędzia do projektowania testów: korzystanie z narzędzi takich jak mapy myśli stało się bardziej popularne z uwagi na możliwość szybkiego projektowania i definiowania testów dla nowej funkcji.
- Narzędzia do zarządzania przypadkami testowymi: narzędzia do zarządzania przypadkami testowymi w projektach zwinnych mogą być częścią narzędzia do zarządzania cyklem życia aplikacji lub narzędzia do zarządzania zadaniami, będącego własnością całego zespołu.
- Narzędzia przygotowujące lub generujące dane testowe: narzędzia, które

generują dane wypełniające bazę danych aplikacji są bardzo korzystne, gdy do przetestowania aplikacji potrzeba wielu danych i ich kombinacji. Narzędzia te mogą również pomóc w ponownym zdefiniowaniu struktury bazy danych, gdy produkt ulega zmianom podczas projektu zwinnego i refaktoryzacji skryptów w celu wygenerowania danych. Pozwala to na szybką aktualizację danych testowych w miarę zachodzących zmian. Niektóre narzędzia do przygotowywania danych testowych wykorzystują źródła danych produkcyjnych jako surowy materiał i używają skryptów do usuwania lub anonimizacji danych wrażliwych. Inne narzędzia do przygotowywania danych testowych mogą pomóc w walidacji dużych zbiorów danych wejściowych lub wyjściowych.

- Narzędzia do ładowania danych testowych: po wygenerowaniu danych do testów należy je załadować do aplikacji. Ręczne wprowadzanie danych jest często pracochłonne i podatne na wprowadzanie błędów, ale dostępne są narzędzia do ładowania danych, które zapewniają, że proces ładowania danych jest niezawodny i wydajny. W rzeczywistości wiele narzędzi do generowania danych zawiera zintegrowany komponent do ładowania danych. W pozostałych przypadkach możliwe jest również zbiorcze ładowanie przy użyciu systemów zarządzania bazami danych.
- Narzędzia do automatycznego wykonywania testów: istnieją narzędzia do wykonywania testów, które z zasady są bardziej dostosowane do testowania zwinnego. 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 pozwalają testerom i biznesowi wyrazić oczekiwane zachowanie systemu w tabelach lub w języku naturalnym za pomocą 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 programisty, ponieważ rejestrują one wykonane czynności. Jest to przydatne w przypadku wykrycia defektu, ponieważ czynności wykonane przed wystąpieniem awarii zostały zarejestrowane i mogą zostać wykorzystane przy raportowaniu defektu programistom. Rejestrowanie kroków wykonywanych podczas sesji testów eksploracyjnych może okazać się korzystne, jeżeli test zostanie ostatecznie włączony do 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 używane są maszyny wirtualne lub instancje
w chmurze, zespoły mają większą liczbę serwerów dostępnych do wytwarzania i testowania. Umożliwia to uniknięcie opóźnienia związanego z oczekiwaniem na fizyczne serwery. Postawienie nowego serwera lub przywrócenie stanu serwera jest bardziej efektywne dzięki funkcjom obrazów stanu wbudowanym w większość narzędzi do wirtualizacji. Niektóre narzędzia do zarządzania testami wykorzystują obecnie technologie wirtualizacji do wykonania zdjęcia serwera w momencie, w którym wykryto

defekt, umożliwiając testerom udostępnienie tego zdjęcia programistom analizującym usterkę.

4 Bibliografia

4.1 Standardy

- [DO-178C] RTCA/FAA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 2012.
- [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 3.1.
- [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 4.0

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, 3rd Edition" Microsoft Press, 2013.

4.4 Terminologia zwinna

Słowa kluczowe, które można znaleźć w Słowniku terminów testowych ISTQB®, zostały wymienione na początku każdego rozdziału. Często spotykane pojęcia zwinne zostały zaczerpnięte z następujących, ogólnie akceptowanych, źródeł internetowych, które podają definicje:
https://www.agilealliance.org/
https://www.techtarget.com/
https://www.scrumalliance.org/

Zachęcamy czytelników do odwołania się do ww. 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][5].

4.5 Inne pozycje

Poniższe odniesienia wskazują na informacje dostępne w Internecie i w innych miejscach. Mimo że odniesienia te zostały sprawdzone w momencie publikacji niniejszego sylabusa, ISTQB® nie może ponosić odpowiedzialności, jeśli odniesienia nie są już dostępne.

- [Agile Alliance Guide] Various contributors, https://guide.agilealliance.org/
- [Agilemanifesto] Various contributors, https://agilemanifesto.org/
- [Hendrickson]: Elisabeth Hendrickson, "Acceptance Test-driven Development," testobsessed.com/2008/12/acceptance-test-driven-development-atdd-an-overview
- [INVEST] Bill Wake, "INVEST in Good Stories, and SMART Tasks," https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

[5] Aktualnie obowiązuje nowa, uproszczona wersja Przewodnika po Scrumie w tłumaczeniu Tomasza Włodarka i innych (2020 rok) https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Polish.pdf

5. Indeks