Certyfikowany tester
Sylabus poziomu podstawowego ISTQB®

Wersja 2018 wersja 3.1

International Software Testing Qualifications Board®

Informacja o prawach autorskich:

Historia zmian polskiej wersji sylabusa

Wersja	Data	Uwagi
3.1	23.02.2021	Przegląd edytorski
3.1	15.11.2019 – 31.03.2020	Aktualizacja do wersji 3.1. – Zespół SJSI
1.1	15.01.2019 – 26.04.2019	Korekta, modyfikacje językowe – Zespół SJSI
1.0	15.07.2018 – 01.09.2018	Przegląd końcowy
0.4	20.06.2018 – 15.07.2018	Aktualizacji wersji 0.3. – Zespół SJSI
	29.05.2018	Zatwierdzenie przez GA ISTQB®
0.3	25.05.2018 – 18.06.2018	Przegląd i wprowadzanie zmian w wersji beta – Zespół SJSI
	01.05.2018	Udostępnienie przez ISTQB® wersji końcowej
0.1	15.04.2018 – 15.05.2018	Przegląd tłumaczenia wersji beta
0.1	01.03.2018 – 12.04.2018	Tłumaczenie wersji beta: BTInfo Biuro Tłumaczeń Informatycznych Przyłuccy Sp. j.

Podziękowania

Dokument ten został formalnie zaakceptowany przez Zgromadzenie Ogólne ISTQB® (11 listopada 2019 r.).

Dokument ten został opracowany przez zespół ISTQB® w składzie: Klaus Olsen (przewodniczący), Meile Posthuma i Stephanie Ulrich.

Zespół dziękuje Radom Członkowskim za komentarze do przeglądu dotyczące Sylabusa Poziomu Podstawowego wersja 2018.

Sylabus Poziomu Podstawowego 2018 został opracowany przez Grupę Roboczą ISTQB® w składzie: Klaus Olsen (przewodniczący), Tauhida Parveen (wiceprzewodnicząca), Rex Black (kierownik projektu), Debra Friedenberg, Judy McKay, Meile Posthuma, Hans Schaefer, Radosław Smilgin, Mike Smith, Stephanie Ulrich, Marie Walsh, Steve Toms i Eshraka Zakaria.

Członkowie zespołu pragną podziękować Rexowi Blackowi i Dorothy Graham za redakcję techniczną oraz zespołowi weryfikatorów, zespołowi ds. weryfikacji krzyżowej i Radom Krajowym za sugestie i wskazówki.

W procesie weryfikacji, zgłaszania uwag i głosowania nad niniejszym sylabusem uczestniczyły następujące osoby (w porządku alfabetycznym): Tom Adams, Tobias Ahlgren, Xu Aiguo, Chris Van Bael, Katalin Balla, Graham Bath, Gualtiero Bazzana, Arne Becher, Veronica Belcher, Lars Hilmar Bjørstrup, Ralf Bongard, Armin Born, Robert Bornelind, Mette Bruhn-Pedersen, Geza Bujdoso, Earl Burba, Filipe Carlos, Young Jae Choi, Greg Collina, Alessandro Collino, Cui Zhe, Taz Daughtrey, Matthias Daigl, Wim Decoutere, Frans Dijkman, Klaudia Dussa-Zieger, Yonit Elbaz, Ofer Feldman, Mark Fewster, Florian Fieber, David Frei, Debra Friedenberg, Conrad Fujimoto, Pooja Gautam, Thorsten Geiselhart, Chen Geng, Christian Alexander Graf, Dorothy Graham, Michel Grandjean, Richard Green, Attila Gyuri, Jon Hagar, Kobi Halperin, Matthias Hamburg, Zsolt Hargitai, Satoshi Hasegawa, Berit Hatten, Wang Hongwei, Tamás Horváth, Leanne Howard, Chinthaka Indikadahena, J. Jayapradeep, Kari Kakkonen, Gábor Kapros, Beata Karpinska, Karl Kemminger, Kwanho Kim, Seonjoon Kim, Cecilia Kjellman, Johan Klintin, Corne Kruger, Gerard Kruijff, Peter Kunit, Hyeyong Kwon, Bruno Legeard, Thomas Letzkus, Alon Linetzki, Balder Lingegård, Tilo Linz, Hongbiao Liu, Claire Lohr, Ine Lutterman, Marek Majernik, Rik Marselis, Romanos Matthaios, Judy McKay, Fergus McLachlan, Dénes Medzihradszky, Stefan Merkel, Armin Metzger, Don Mills, Gary Mogyorodi, Ninna Morin, Ingvar Nordström, Adam Novak, Avi Ofer, Magnus C. Ohlsson, Joel Oliviera, Monika Stocklein Olsen, Kenji Onishi, Francisca Cano Ortiz, Gitte Ottosen, Tuula Pääkkönen, Ana Paiva, Tal Pe'er, Helmut Pichler, Michaël Pilaeten, Horst Pohlmann, Andrew Pollner, Meile Posthuma, Vitalijs Puiso, Salvatore Reale, Stuart Reid, Ralf Reissing, Shark Ren, Miroslav Renda, Randy Rice, Adam Roman, Jan Sabak, Hans Schaefer, Ina Schieferdecker, Franz Schiller, Jianxiong Shen, Klaus Skafte, Mike Smith, Cristina Sobrero, Marco Sogliani, Murian Song, Emilio Soresi, Helder Sousa, Michael Sowers, Michael Stahl, Lucjan Stapp, Li Suyuan, Toby Thompson, Steve Toms, Sagi Traybel, Sabine Uhde, Stephanie Ulrich, Philippos Vakalakis, Erik van Veenendaal, Marianne Vesterdal, Ernst von Düring, Salinda Wickramasinghe, Marie Walsh, Søren Wassard, Hans Weiberg, Paul Weymouth, Hyungjin Yoon, John Young, Surong Yuan, Ester Zabar oraz Karolina Zmitrowicz.

W czasie, gdy tworzony był Sylabus Poziomu Podstawowego ISTQB® 2018, w skład Grupy Roboczej Poziomu Podstawowego wchodziły następujące osoby: Klaus Olsen (przewodniczący), Tauhida Parveen (wiceprzewodnicząca), Rex Black (kierownik projektu), Dani Almog, Debra Friedenberg, Rashed Karim, Johan Klintin, Vipul Kocher, Corne Kruger, Sunny Kwon, Judy McKay, Thomas Müller, Igal Levi, Ebbe Munk, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Hans Schaefer, Radosław Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria oraz Stevan Zivanovic. Grupa Robocza Poziomu Podstawowego dziękuje zespołowi recenzentów i wszystkim Radom Krajowym za ich sugestie.

Grupa Robocza Poziomu Podstawowego, wersja 2011: Thomas Müller (przewodniczący), Debra Friedenberg. Grupa Robocza Poziomu Podstawowego dziękuje zespołowi recenzentów (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal) i wszystkim Radom Krajowym za ich sugestie.

Grupa Robocza Poziomu Podstawowego, wersja 2011: Thomas Müller (przewodniczący), Rahul Verma, Martin Klonk oraz Armin Beer. Grupa Robocza Poziomu Podstawowego dziękuje zespołowi recenzentów (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal) i wszystkim Radom Krajowym za ich sugestie.

Grupa Robocza Poziomu Podstawowego, wersja 2007: Thomas Müller (przewodniczący), Dorothy Graham, Debra Friedenberg i Erik van Veenendaal. Grupa Robocza Poziomu Podstawowego dziękuje zespołowi recenzentów (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, oraz Wonil Kwon) i wszystkim Radom Krajowym za ich sugestie.

Grupa Robocza Poziomu Podstawowego, wersja 2007: Thomas Müller (przewodniczący), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson oraz Erik van Veenendaal. Grupa Robocza Poziomu Podstawowego dziękuje zespołowi recenzentów i wszystkim Radom Krajowym za ich sugestie.

0. Wstęp

0.1. Cel niniejszego sylabusa

Niniejszy sylabus stanowi podstawę egzaminu International Software Testing Qualifications Board na poziomie podstawowym. ISTQB® udostępnia sylabus:

1. Radom Krajowym w celu przetłumaczenia na języki lokalne i dokonania akredytacji dostawców szkoleń (Rady Krajowe mogą dostosowywać sylabus do potrzeb danego języka oraz dodawać odwołania do literatury w celu uwzględnienia publikacji lokalnych);
2. organom certyfikacyjnym w celu sformułowania pytań egzaminacyjnych w językach lokalnych (dostosowanych do celów nauczania niniejszego sylabusa);
3. dostawcom szkoleń w celu opracowania materiałów dydaktycznych i określenia odpowiednich metod nauczania;
4. kandydatom ubiegającym się o certyfikat w celu przygotowania się do egzaminu certyfikacyjnego (w ramach szkoleń lub samodzielnie);
5. międzynarodowej społeczności specjalistów w dziedzinie inżynierii oprogramowania i systemów informatycznych w celu rozwijania zawodu testera oprogramowania i systemów informatycznych oraz opracowywania książek i artykułów.

ISTQB® może również zezwolić innym podmiotom na korzystanie z niniejszego sylabusa do innych celów pod warunkiem wystąpienia przez te podmioty o stosowną pisemną zgodę do ISTQB®.

0.2. Certyfikowany tester — poziom podstawowy w testowaniu oprogramowania

Kwalifikacja na poziomie podstawowym jest przeznaczona dla wszystkich osób zaangażowanych w testowanie oprogramowania. Mogą to być między innymi: testerzy, analitycy testów, inżynierowie testów, konsultanci ds. testów, kierownicy testów, użytkownicy wykonujący testowanie akceptacyjne oraz programiści. Ponadto kwalifikacja na poziomie podstawowym jest odpowiednia dla osób chcących zdobyć podstawową wiedzę w dziedzinie testowania oprogramowania, takich jak: właściciele produktu, kierownicy projektu, kierownicy ds. jakości, kierownicy ds. wytwarzania oprogramowania, analitycy biznesowi, dyrektorzy ds. informatyki oraz konsultanci w dziedzinie zarządzania. Posiadacze certyfikatu podstawowego mogą zdobywać wyższe poziomy kwalifikacji w procesie certyfikacji w dziedzinie testowania oprogramowania.

„ISTQB® Przegląd Poziomu Podstawowego 2018” to oddzielny dokument, który jest ważny także dla Sylabusa Poziomu Podstawowego 2018 v 1.3. i w którym zawarto następujące informacje:

- cele biznesowe sylabusa;
- macierz powiązań między celami biznesowymi a celami nauczania;
- podsumowanie niniejszego sylabusa.

0.3. Cele nauczania objęte egzaminem i poziomy wiedzy

Cele nauczania wspomagają osiąganie celów biznesowych i stanowią wytyczne do tworzenia egzaminów certyfikacyjnych dla testerów na poziomie podstawowym.

Co do zasady przedmiotem egzaminu może być cała treść niniejszego sylabusa na poziomie K1, z wyjątkiem wstępu i załączników. Oznacza to, że od kandydata może być wymagane rozpoznanie, zapamiętanie lub przypomnienie sobie słowa kluczowego lub pojęcia wymienionego w każdym z sześciu rozdziałów dokumentu. Poziomy wiedzy dla poszczególnych celów nauczania przedstawiono na początku każdego rozdziału. Poziomy te sklasyfikowano następująco:

- K1 — zapamiętać;
- K2 — zrozumieć;
- K3 — zastosować.

Dalsze szczegóły i przykłady celów nauczania podano w Załączniku B.

Definicje wszystkich pojęć wymienionych jako słowa kluczowe pod tytułem rozdziałów należy zapamiętać (K1), nawet jeśli nie wspomniano o tym wyraźnie w celach nauczania.

0.4. Egzamin certyfikacyjny na poziomie podstawowym

Egzamin umożliwiający uzyskanie certyfikatu na poziomie podstawowym bazuje na niniejszym sylabusie. Przy udzielaniu odpowiedzi na pytania egzaminacyjne może być konieczne skorzystanie z materiału obejmującego więcej niż jeden rozdział tego sylabusa. Przedmiotem egzaminu może być treść wszystkich części sylabusa z wyjątkiem wstępu i załączników. W dokumencie znajdują się również odwołania do norm/standardów, książek i innych sylabusów ISTQB®, ale ich treść nie może być przedmiotem egzaminu w zakresie wykraczającym poza informacje streszczone w tym sylabusie.

Egzamin ma formę testu wielokrotnego wyboru i składa się z 40 pytań. Do zdania egzaminu niezbędne jest udzielenie poprawnej odpowiedzi na co najmniej 65% pytań (tj. 26 pytań).

Egzaminy można zdawać w ramach akredytowanego szkolenia lub samodzielnie (np. w ośrodku egzaminacyjnym lub w ramach egzaminu publicznego). Ukończenie akredytowanego kursu nie jest warunkiem koniecznym przystąpienia do egzaminu.

0.5. Akredytacja

Rada Krajowa ISTQB® może dokonywać akredytacji dostawców szkoleń, którzy oferują materiały dydaktyczne zgodne z niniejszym sylabusem. Wytyczne dotyczące akredytacji należy uzyskać od Rady Krajowej lub organu dokonującego akredytacji. Akredytowany kurs jest uznawany za zgodny z niniejszym sylabusem i może obejmować egzamin ISTQB®.

0.6. Szczegółowość

Szczegółowość informacji zawartych w niniejszym sylabusie umożliwia tworzenie spójnych pod względem treści nauczania kursów i przeprowadzanie egzaminów na skalę międzynarodową. Aby sprostać temu zadaniu w sylabusie uwzględniono:

- ogólne cele dydaktyczne opisujące założenia poziomu podstawowego;
- wykaz terminów, które muszą zapamiętać uczestnicy szkolenia;
- cele nauczania w poszczególnych obszarach wiedzy, opisujące efekty kształcenia o charakterze poznawczym;
- opis najważniejszych pojęć, w tym odsyłacze do źródeł (takich jak uznane publikacje oraz normy i standardy).

Treść sylabusa nie stanowi opisu całego obszaru wiedzy związanego z testowaniem oprogramowania. Odzwierciedla ona jedynie poziom szczegółowości, jaki należy uwzględnić w akredytowanych szkoleniach na poziomie podstawowym. W sylabusie skupiono się na pojęciach i technikach związanych z testowaniem, które mogą mieć zastosowanie do wszystkich projektów wytwarzania oprogramowania, w tym projektów programowania zwinnego.

Sylabus nie zawiera żadnych konkretnych celów nauczania związanych z określonym cyklem życia oprogramowania lub z określoną metodą wytwarzania oprogramowania, natomiast omówiono w nim, w jaki sposób wprowadzone pojęcia można zastosować do modelu zwinnego wytwarzania oprogramowania, do innych modeli iteracyjnych i przyrostowych oraz do modeli sekwencyjnych.

0.7. Struktura sylabusa

Dokument podzielono na sześć głównych rozdziałów. Nagłówek najwyższego poziomu zawiera informację o czasie trwania każdego rozdziału (nie podano czasu trwania podrozdziałów i mniejszych jednostek redakcyjnych). W przypadku akredytowanych szkoleń na przekazanie wiedzy zawartej w sylabusie potrzeba co najmniej 16,75 godz. wykładu. Czas ten podzielono na poszczególne rozdziały w następujący sposób:

- Rozdział 1. Podstawy testowania — 175 min.
- Rozdział 2. Testowanie w cyklu życia oprogramowania — 100 min.
- Rozdział 3. Testowanie statyczne — 135 min.
- Rozdział 4. Techniki testowania — 330 min.
- Rozdział 5. Zarządzanie testami — 225 min.
- Rozdział 6. Narzędzia wspomagające testowanie — 40 min.

1. Podstawy testowania — 175 min.

Słowa kluczowe

analiza testów, awaria, cel testów, dane testowe, debugowanie, defekt, implementacja testów, jakość, monitorowanie testów i nadzór nad testami, planowanie testów, podstawa testów, podstawowa przyczyna, pokrycie, pomyłka, procedura testowa, proces testowy, projektowanie testów, przedmiot testów, przypadek testowy, śledzenie, testalia, testowanie, ukończenie testów, walidacja, warunek testowy, weryfikacja, wykonywanie testów, wyrocznia testowa, zapewnienie jakości, zestaw testowy

Podstawy testowania — cele nauczania

1.1. Co to jest testowanie?

FL-1.1.1. (K1) Kandydat potrafi wskazać typowe cele testowania.

FL-1.1.2. (K2) Kandydat potrafi odróżnić testowanie od debugowania.

1.2. Dlaczego testowanie jest niezbędne?

FL-1.2.1. (K2) Kandydat potrafi podać przykłady wskazujące, dlaczego testowanie jest niezbędne.

FL-1.2.2. (K2) Kandydat potrafi opisać relację między testowaniem a zapewnieniem jakości oraz podać przykłady wskazujące, w jaki sposób testowanie przyczynia się do podnoszenia jakości.

FL-1.2.3. (K2) Kandydat potrafi rozróżnić pomyłkę, defekt i awarię.

FL-1.2.4. (K2) Kandydat potrafi odróżnić podstawową przyczynę od skutków defektu.

1.3. Siedem zasad testowania

FL-1.3.1. (K2) Kandydat potrafi objaśnić siedem zasad testowania.

1.4. Proces testowy

FL-1.4.1. (K2) Kandydat potrafi wyjaśnić wpływ kontekstu na proces testowy.

FL-1.4.2. (K2) Kandydat potrafi opisać czynności testowe i odpowiadające im zadania w ramach procesu testowego.

FL-1.4.3. (K2) Kandydat potrafi rozróżnić produkty pracy wspomagające proces testowy.

FL-1.4.4. (K2) Kandydat potrafi wyjaśnić korzyści wynikające ze śledzenia powiązań między podstawą testów a produktami pracy związanymi z testowaniem.

1.5. Psychologia testowania

FL-1.5.1. (K1) Kandydat potrafi wskazać czynniki psychologiczne wpływające na powodzenie testowania.

FL-1.5.2. (K2) Kandydat potrafi wyjaśnić różnice w sposobie myślenia testerów i programistów.

1.1. Co to jest testowanie?

Systemy oprogramowania są nieodłączną częścią naszego życia we wszystkich jego obszarach — od aplikacji biznesowych (np. w bankowości) po produkty użytkowe (np. samochody). Jednocześnie większość z nas miała zapewne do czynienia z oprogramowaniem, które nie zadziałało tak, jak powinno. Nieprawidłowe funkcjonowanie oprogramowania może powodować wiele problemów, w tym straty finansowe, stratę czasu, utratę reputacji firmy, a nawet utratę zdrowia lub życia. Testowanie oprogramowania pozwala ocenić jego jakość i zmniejszyć ryzyko wystąpienia awarii podczas eksploatacji.

Powszechnie uważa się, że testowanie polega wyłącznie na wykonywaniu testów, czyli uruchamianiu oprogramowania i sprawdzaniu uzyskanych rezultatów. Jak jednak opisano w podrozdziale 1.4., testowanie oprogramowania to proces obejmujący czynności, które wykraczają poza samo wykonywanie testów. W skład procesu testowego wchodzą również takie czynności jak: planowanie, analiza, monitorowanie testów i nadzór nad testami, projektowanie i implementacja testów, raportowanie o postępie i wynikach testów oraz dokonywanie oceny jakości przedmiotu testów. Testowanie może wymagać uruchomienia testowanego modułu lub systemu – mamy wtedy do czynienia z tzw. testowaniem dynamicznym. Można również wykonywać testy bez uruchamiania testowanego obiektu – takie testowanie nazywa się testowaniem statycznym, a zatem testowanie obejmuje również przegląd produktów pracy takich jak: wymagania, historyjki użytkownika i kod źródłowy.

Inne nieporozumienie polega na postrzeganiu testowania jako czynności skupionej wyłącznie na weryfikacji wymagań, historyjek użytkownika lub innych form specyfikacji. Chociaż w ramach testowania rzeczywiście sprawdza się, czy system spełnia wyspecyfikowane wymagania, to jednak przeprowadza się również walidację, której zadaniem jest sprawdzenie, czy system odpowiada na potrzeby użytkowników oraz innych interesariuszy w swoim środowisku operacyjnym.

Czynności testowe są organizowane i przeprowadzane w różny sposób w różnych cyklach życia oprogramowania (patrz podrozdział 2.1.).

1.1.1. Typowe cele testowania

W przypadku każdego projektu cele testowania mogą być realizowane między innymi poprzez następujące czynności:

- zapobieganie defektom poprzez dokonywanie oceny produktów pracy, takich jak: wymagania, historyjki użytkownika, projekt i kod;
- weryfikacja, czy zostały spełnione wszystkie wyspecyfikowane wymagania;
- sprawdzanie, czy przedmiot testów jest kompletny i walidowanie, czy działa zgodnie z oczekiwaniami użytkowników i innych interesariuszy;
- budowanie zaufania do poziomu jakości przedmiotu testów;
- wykrywanie defektów i awarii, a tym samym zmniejszenie poziomu ryzyka związanego z niedostateczną jakością oprogramowania;
- dostarczanie interesariuszom informacji niezbędnych do podejmowania świadomych decyzji (dotyczących zwłaszcza poziomu jakości przedmiotu testów);
- przestrzeganie wymagań wynikających z umów, przepisów prawa i norm/standardów i/lub sprawdzanie, czy obiekt testów jest zgodny z tymi wymaganiami lub standardami.

Cele testowania mogą różnić się w zależności od kontekstu testowanego modułu lub systemu, poziomu testów oraz modelu cyklu życia oprogramowania. Poniżej podano kilka przykładowych różnic.

- W przypadku testowania modułowego jednym z celów może być wykrycie jak największej liczby awarii, a w rezultacie wczesne zidentyfikowanie i usunięcie powodujących je defektów. Innym celem może być zwiększenie pokrycia kodu przez testy modułowe.
- W przypadku testowania akceptacyjnego jednym z celów może być potwierdzenie, że system działa zgodnie z oczekiwaniami i spełnia stawiane mu wymagania. Innym celem tego rodzaju testowania może być dostarczenie interesariuszom informacji na temat ryzyka, jakie wiąże się z przekazaniem systemu do eksploatacji w danym momencie.

1.1.2. Testowanie a debugowanie

Testowanie i debugowanie to dwie różne czynności. Wykonywanie testów pozwala ujawnić awarie, które są skutkiem defektów w oprogramowaniu, natomiast debugowanie to czynność związana z wytwarzaniem oprogramowania, która polega na znajdowaniu, analizowaniu i usuwaniu tych defektów. Następnie wykonywane jest testowanie potwierdzające, które pozwala sprawdzić, czy wprowadzone poprawki w rezultacie spowodowały usunięcie defektów. W niektórych przypadkach testerzy są odpowiedzialni za przeprowadzenie testu początkowego oraz końcowego testu potwierdzającego, a programiści są odpowiedzialni za przeprowadzenie debugowania odpowiedniego modułu oraz testowania integracyjnego modułów (ciągła integracja). Jednakże w przypadku zwinnego wytwarzania oprogramowania i niektórych innych cykli życia wytwarzania oprogramowania testerzy mogą być również zaangażowani w debugowanie i testowanie modułowe.

Standard międzynarodowy ISO/IEC/IEEE 29119-1 zawiera bardziej szczegółowe informacje dotyczące pojęć związanych z testowaniem.

1.2. Dlaczego testowanie jest niezbędne?

Rygorystyczne testowanie modułów i systemów oraz związanej z nimi dokumentacji może pomóc w zmniejszeniu ryzyka wystąpienia awarii podczas eksploatacji oprogramowania. Wykrycie, a następnie usunięcie defektów, przyczynia się do podniesienia jakości modułów lub systemów. Ponadto testowanie oprogramowania może być niezbędne do spełnienia wymagań wynikających z umów, przepisów prawa bądź norm/standardów branżowych.

1.2.1. Znaczenie testowania dla powodzenia projektu

W historii informatyki znanych jest wiele przykładów oprogramowania i systemów, które zostały przekazane do eksploatacji, ale na skutek defektów uległy awarii lub z innych powodów nie zaspokoiły potrzeb interesariuszy. Dzięki odpowiednim technikom testowania — stosowanym w sposób fachowy na odpowiednich poziomach testów i w odpowiednich fazach cyklu życia oprogramowania — częstotliwość występowania tego rodzaju problemów można jednak ograniczyć. Poniżej przedstawiono kilka przykładów takich zastosowań odpowiednich technik testowania.

- Zaangażowanie testerów w przeglądy wymagań lub w doprecyzowywanie historyjek użytkownika pomaga wykryć defekty w tych produktach pracy. Zidentyfikowanie i usunięcie tych defektów zmniejsza ryzyko wytworzenia niepoprawnych lub nietestowalnych cech oprogramowania.
- Ścisła współpraca między testerami a projektantami systemu na etapie prac projektowych pozwala obu stronom lepiej zrozumieć projekt systemu i sposób jego testowania. Lepsza znajomość tematu przekłada się na mniejsze ryzyko wystąpienia zasadniczych problemów w projekcie, a także pozwala zidentyfikować testy w początkowym etapie projektu.

- Ścisła współpraca testerów z programistami na etapie tworzenia kodu pozwala obu stronom lepiej zrozumieć kod i sposób jego testowania. Większa wiedza w tym zakresie pozwala zmniejszyć ryzyko wystąpienia defektów w kodzie i spowodowanych przez nie awarii w testach.
- Weryfikacja i walidacja oprogramowania przez testerów przed przekazaniem go do eksploatacji pozwala wykryć defekty mogące prowadzić do awarii, które w przeciwnym razie mogłyby zostać przeoczone. Ułatwia także usuwanie związanych z nimi (awariami) defektów (debugowanie). W rezultacie rośnie prawdopodobieństwo, że oprogramowanie zaspokoi potrzeby interesariuszy i spełni stawiane mu wymagania.

Aby uzupełnić powyższe przykłady warto dodać, że osiągnięcie zdefiniowanych celów testowania (patrz p. 1.1.1.) przyczynia się do ogólnego powodzenia procesu wytwarzania i pielęgnacji oprogramowania.

1.2.2. Zapewnienie jakości a testowanie

Testowanie jest często utożsamiane z zapewnieniem jakości (ang. *quality assurance*), ale w rzeczywistości są to dwa oddzielne (chociaż powiązane ze sobą) procesy, które zawierają się w szerszym pojęciu „zarządzania jakością" (ang. *quality management*). Zarządzanie jakością obejmuje wszystkie czynności mające na celu kierowanie i nadzorowanie działań organizacji w dziedzinie jakości. Elementami zarządzania jakością są między innymi zapewnienie jakości i kontrola jakości. Zapewnienie jakości skupia się zazwyczaj na skrupulatnym realizowaniu właściwych procesów w celu uzyskania pewności, że zostaną osiągnięte odpowiednie poziomy jakości. Jeśli bowiem procesy są realizowane prawidłowo, powstające w ich wyniku produkty pracy mają zwykle wyższą jakość, a to przyczynia się do zapobiegania defektom. Duże znaczenie dla skutecznego zapewnienia jakości mają również wykorzystanie procesu analizy przyczyny podstawowej do wykrywania i usuwania przyczyn defektów oraz prawidłowe wdrażanie wniosków ze spotkań retrospektywnych w celu doskonalenia procesów.

Kontrola jakości obejmuje cały szereg czynności, włączając w to czynności testowe, które wspierają osiągnięcie odpowiednich poziomów jakości w procesie wytwarzania oprogramowania. Czynności testowe są ważnym elementem procesu wytwarzania lub pielęgnacji oprogramowania. Prawidłowy przebieg tego procesu (w tym testowania) jest istotny z punktu widzenia zapewnienia jakości, w związku z czym działania związane z zapewnieniem jakości wspierają właściwe testowanie. Jak opisano w punktach 1.1.1. i 1.2.1., testowanie przyczynia się do osiągnięcia wymaganej jakości produktów pracy na kilka różnych sposobów.

1.2.3. Pomyłki, defekty i awarie

Na skutek pomyłki (błędu) człowieka w kodzie oprogramowania lub w innym związanym z nim produkcie pracy może powstać defekt (inaczej zwany usterką lub pluskwą). Pomyłka skutkująca wprowadzeniem defektu w jednym produkcie pracy może spowodować błąd skutkujący wprowadzeniem defektu w innym, powiązanym produkcie pracy. Przykładem takiej sytuacji jest pomyłka popełniona podczas pozyskiwania wymagań, która może prowadzić do defektu w wymaganiach, co spowoduje pomyłkę programisty skutkującą wprowadzeniem defektu w kodzie.

Wykonanie kodu zawierającego defekt może spowodować awarię, ale nie musi dziać się tak w przypadku każdego defektu. Niektóre defekty powodują awarię na przykład tylko po wprowadzeniu ściśle określonych danych wejściowych bądź na skutek wystąpienia określonych warunków wstępnych, które mogą zaistnieć bardzo rzadko lub nigdy.

Pomyłki mogą pojawiać się z wielu powodów, takich jak:

- presja czasu;

- omylność człowieka;
- brak doświadczenia lub niedostateczne umiejętności uczestników projektu;
- problemy związane z wymianą informacji między uczestnikami projektu (w tym nieporozumienia dotyczące rozumienia wymagań i dokumentacji projektowej);
- złożoność kodu, projektu, architektury, rozwiązywanego problemu i/lub wykorzystywanej technologii;
- nieporozumienia dotyczące interfejsów wewnątrz systemu i między systemami, zwłaszcza w przypadku dużej liczby tych systemów;
- stosowanie nowych, nieznanych technologii.

Awarie – poza tymi wynikającymi z defektów w kodzie – mogą być również spowodowane warunkami środowiskowymi. Przykładem takich sytuacji i warunków są: promieniowanie, pole elektromagnetyczne lub zanieczyszczenie środowiska, które mogą spowodować wystąpienie defektów w oprogramowaniu wewnętrznym (ang. *firmware*) lub wpłynąć na działanie oprogramowania poprzez zmianę parametrów pracy sprzętu.

Nie wszystkie nieoczekiwane wyniki testów oznaczają awarie. Wynik fałszywie pozytywny może być, między innymi, skutkiem błędów związanych z wykonaniem testów, defektów w danych testowych, środowisku testowym, w innych testaliach itp. Podobne problemy mogą być przyczyną sytuacji odwrotnej – wyniku fałszywie negatywnego, czyli sytuacji, w której testy nie wykrywają defektu, który powinny wykryć. Wyniki fałszywie pozytywne są raportowane jako defekty, których w rzeczywistości nie ma.

1.2.4. Defekty, podstawowe przyczyny oraz skutki

Podstawowa przyczyna defektu to najwcześniejsze czynności lub warunki, które przyczyniły się do powstania defektuł. Przeanalizowanie defektu w celu zidentyfikowania podstawowej przyczyny pozwala zredukować wystąpienia podobnych defektów w przyszłości. Ponadto analiza przyczyny podstawowej — skupiająca się na najważniejszych, pierwotnych przyczynach defektów — może prowadzić do udoskonalenia procesów, co może z kolei przełożyć się na dalsze zmniejszenie liczby defektów w przyszłości.

Załóżmy na przykład, że defekt w jednym z wierszy kodu powoduje nieprawidłowe wypłacanie odsetek, co z kolei wiąże się z reklamacjami klientów. Wadliwy kod został napisany na podstawie historyjki użytkownika, która była niejednoznaczna, ponieważ właściciel produktu źle zrozumiał zasady naliczania odsetek. W związku z tym, jeśli przy naliczaniu odsetek występuje duży procent defektów, których podstawową przyczyną są podobne nieporozumienia, rozwiązaniem może być przeszkolenie właścicieli produktów w zakresie tego rodzaju obliczeń, co pozwoli zmniejszyć w przyszłości liczbę podobnych defektów.

W powyższym przykładzie reklamacje klientów są skutkami, nieprawidłowa wypłata odsetek jest awarią, a nieprawidłowe obliczenia wykonywane przez kod są defektem wynikającym z pierwotnego defektu, jakim była niejednoznaczność historyjki użytkownika. Podstawową przyczyną pierwotnego defektu były braki wiedzy właściciela produktu, na skutek których popełnił on pomyłkę przy pisaniu historyjki użytkownika. Proces analizy przyczyny podstawowej omówiono w sylabusach [ISTQB CTEL_TM] i [ISTQB EITP].

1.3. Siedem zasad testowania

Przez ostatnie kilkadziesiąt lat zaproponowano cały szereg zasad testowania, które dostarczają ogólnych wskazówek mających zastosowanie do wszystkich rodzajów testowania.

1. Testowanie ujawnia usterki, ale nie może dowieść ich braku

Testowanie może wykazać obecność defektów, ale nie może dowieść, że oprogramowanie jest od nich wolne. Tym samym testowanie zmniejsza prawdopodobieństwo, że w oprogramowaniu pozostaną niewykryte defekty, ale sam fakt niewykrycia defektów nie stanowi dowodu poprawności oprogramowania.

2. Testowanie gruntowne jest niemożliwe

Przetestowanie wszystkiego (tj. wszystkich kombinacji danych wejściowych i warunków wstępnych) jest możliwe tylko w najprostszych przypadkach. W związku z tym, wysiłki w testowaniu powinny być ukierunkowane raczej na analizę ryzyka, techniki testowania i priorytetyzację niż dążenie do testowania.

3. Wczesne testowanie oszczędza czas i pieniądze

Aby wcześnie wykryć defekty, należy rozpocząć testowanie statyczne i dynamiczne na jak najwcześniejszym etapie cyklu życia oprogramowania. Wczesne testowanie jest niekiedy nazywane „przesunięciem w lewo” (ang. *shift left*). Wykonywanie testów na wczesnym etapie cyklu życia oprogramowania pozwala ograniczyć lub wyeliminować kosztowne zmiany (patrz podrozdział 3.1.).

4. Kumulowanie się defektów

Zwykle większość defektów wykrytych podczas testowania przed przekazaniem oprogramowania do eksploatacji lub większość awarii występujących w fazie eksploatacji występuje lub ma swoje źródło w niewielkiej liczbie modułów. W rezultacie przewidywane skupiska defektów i skupiska defektów faktycznie zaobserwowane na etapie testowania lub eksploatacji są ważnym elementem analizy ryzyka, którą przeprowadza się w celu odpowiedniego ukierunkowania wysiłków związanych z testowaniem (o czym wspomniano w zasadzie nr 2.).

5. Paradoks pestycydów

Ciągłe powtarzanie tych samych testów prowadzi do sytuacji, w której przestają one w pewnym momencie wykrywać nowe defekty. Aby móc wykrywać nowe defekty, może być konieczne zmodyfikowanie dotychczasowych testów i danych testowych, a także napisanie nowych testów. Niezmieniane testy tracą z czasem zdolność do wykrywania defektów, podobnie jak pestycydy po pewnym czasie nie są zdolne do eliminowania szkodników. W niektórych przypadkach — takich jak automatyczne testowanie regresji — paradoks pestycydów może być korzystny, ponieważ pozwala potwierdzić, że liczba defektów związanych z regresją jest niewielka.

6. Testowanie zależy od kontekstu

Testowanie wykonuje się w różny sposób w różnych kontekstach. Na przykład oprogramowanie sterujące systemami przemysłowymi, które jest krytyczne ze względów bezpieczeństwa, testuje się inaczej niż aplikację mobilną sklepu internetowego. Innym przykładem może być odmienny sposób przeprowadzania testów w projektach zwinnych i w tych prowadzonych zgodnie z modelem sekwencyjnym cyklu życia wytwarzania oprogramowania (patrz podrozdział 2.1.).

7. Przekonanie o braku błędów jest błędem[1]

Niektóre organizacje oczekują, że testerzy będą w stanie uruchomić wszystkie możliwe testy i wykryć wszystkie możliwe defekty, ale powyższe zasady (odpowiednio nr 2 i 1) pokazują, że jest to niemożliwe. Co więcej, błędnym jest przekonanie, że samo znalezienie i naprawienie dużej liczby defektów zapewni pomyślne wdrożenie systemu (na przykład bardzo dokładne testowanie wszystkich wyspecyfikowanych wymagań i naprawienie wszystkich znalezionych defektów wciąż może nas nie uchronić od zbudowania systemu trudnego w obsłudze, który nie zaspokoi potrzeb oraz nie spełni oczekiwań użytkowników lub będzie miał gorsze parametry od konkurencyjnych rozwiązań).

Więcej informacji na temat powyższych zagadnień oraz wielu innych zasad testowania można znaleźć w [Myers 2011], [Kaner 2002], [Weinberg 2008] oraz [Beizer 1990].

1.4. Proces testowy

Nie ma jednego uniwersalnego procesu testowania oprogramowania, ale istnieją typowe czynności testowe, bez stosowania których nie zostaną zrealizowane ustalone dla testowania cele. Czynności te składają się na proces testowy. Dobór procesu testowego do konkretnego oprogramowania w konkretnej sytuacji zależy od wielu czynników. Organizacja może w swojej strategii testowej zdefiniować, które czynności testowe wchodzą w skład tego procesu, jak czynności testowe mają być zaimplementowane oraz w którym momencie cyklu życia oprogramowania powinny mieć miejsce.

1.4.1. Proces testowy w kontekście

Przykładowe czynniki kontekstowe wpływające na proces testowy w organizacji to:

- wykorzystywany model cyklu życia oprogramowania i metodyki projektowe;
- rozważane poziomy testów i typy testów;
- czynniki ryzyka produktowego i projektowego;
- dziedzina biznesowa;
- ograniczenia operacyjne, w tym w szczególności:
  - budżety i zasoby;
  - harmonogramy;
  - złożoność procesu lub projektu;
  - wymagania wynikające z umów i przepisów;
- polityka testów i praktyki obowiązujące w organizacji;
- wymagane normy/standardy wewnętrzne i zewnętrzne.

W kolejnych punktach opisano ogólne aspekty organizacyjnego procesu testowego w relacji do:

- czynności i zadań testowych;
- produktów pracy związanych z testowaniem;

[1] Ta zasada powinna się nazywać „Błędne przekonanie o braku defektów", tym niemniej zdecydowaliśmy się zostawić „Przekonanie o braku błędów jest błędem" jako najbliższe angielskiemu „*Absence-of-errors fallacy*".

- możliwości śledzenia powiązań pomiędzy podstawą testów a produktami pracy związanymi z testowaniem.

Dobrą praktyką jest zdefiniowanie mierzalnych kryteriów pokrycia dotyczących podstawy testów (w odniesieniu do każdego rozważanego poziomu lub typu testów). Kryteria pokrycia mogą w praktyce pełnić funkcję kluczowych wskaźników wydajności (ang *Key Performance Indicators* — KPI) sprzyjających wykonywaniu określonych czynności i pozwalających wykazać osiągnięcie celów testowania oprogramowania (patrz p. 1.1.1.).

Przykładem ilustrującym tę praktykę jest podstawa testów dla aplikacji mobilnej, która może zawierać listę wymagań oraz listę wspieranych urządzeń mobilnych. Każde wymaganie i każde wspierane urządzenie jest elementem podstawy testów. Kryteria pokrycia mogą wymagać co najmniej jednego przypadku testowego dla każdego elementu podstawy testów. Po wykonaniu testów ich wyniki są źródłem informacji dla interesariuszy, czy określone wymagania zostały spełnione oraz czy na wspieranych urządzeniach zaobserwowano awarie.

Więcej informacji o procesach testowych można znaleźć w standardzie ISO [ISO/IEC/IEEE 29119-2].

1.4.2. Czynności i zadania testowe

W procesie testowym wyróżnia się następujące, główne grupy czynności:

- planowanie testów;
- monitorowanie testów i nadzór nad testami;
- analiza testów;
- projektowanie testów;
- implementacja testów;
- wykonywanie testów;
- ukończenie testów.

Każda główna grupa składa się z czynności, które opisano w kolejnych podpunktach. Ponadto poszczególne czynności w każdej z grup mogą składać się z kilku pojedynczych zadań różniących się w zależności od projektu lub wersji oprogramowania.

Należy również zaznaczyć, że chociaż wiele głównych grup czynności może sprawiać wrażenie logicznie uszeregowanych, często są one realizowane metodą iteracyjną. Przykładem takiej praktyki jest model zwinnego wytwarzania oprogramowania, który opiera się na krótkich iteracjach obejmujących projektowanie, tworzenie i testowanie produktu. Iteracje odbywają się w sposób ciągły i są objęte ciągłym planowaniem. W rezultacie czynności testowe są również wykonywane w ramach tego podejścia do wytwarzania oprogramowania w sposób ciągły i iteracyjny. Nawet w przypadku stosowania modeli sekwencyjnych wytwarzania oprogramowania, w których główne czynności są wykonywane krokowo w logicznej kolejności, pewne elementy nakładają się na siebie, są wykonywane łącznie lub równocześnie bądź są pomijane. W związku z powyższym zwykle konieczne jest dostosowanie głównych grup czynności do kontekstu danego systemu lub projektu.

Planowanie testów

Planowanie testów obejmuje czynności, których efektem jest zdefiniowanie celów testowania oraz określenie podejścia do osiągania celów testowania w granicach wyznaczonych przez kontekst (np. określenie odpowiednich technik testowania i zadań testowych oraz sformułowanie harmonogramu testów, który umożliwi dotrzymanie wyznaczonego terminu). Plany testów mogą być następnie korygowane na podstawie informacji zwrotnych z monitorowania i nadzoru. Kwestię planowania testów objaśniono dokładniej w podrozdziale 5.2.

Monitorowanie testów i nadzór nad testami

Monitorowanie testów polega na ciągłym porównywaniu rzeczywistego z zaplanowanym postępem testowania przy użyciu miar specjalnie w tym celu zdefiniowanych w planie testów. Nadzór nad testami polega na podejmowaniu działań, które są niezbędne do osiągnięcia celów wyznaczonych w planie testów (z uwzględnieniem jego ewentualnych aktualizacji). Elementem wspomagającym monitorowanie testów i nadzór nad testami jest ocena kryteriów wyjścia, które w przypadku niektórych modeli cykli życia wytwarzania oprogramowania są również nazywane „definicją ukończenia” (ang. *Definition of Done*; patrz [ISTQB-CTEL-AT]). Ocena kryteriów wyjścia dla wykonania testów na określonym poziomie testów może obejmować:

- sprawdzenie rezultatów testów i dziennika testów pod kątem określonych kryteriów pokrycia;
- oszacowanie poziomu jakości modułu lub systemu na podstawie rezultatów testów i dziennika testów;
- ustalenie, czy są konieczne dalsze testy (np. w przypadku nieosiągnięcia przez dotychczas wykonane testy pierwotnie założonego poziomu pokrycia ryzyka produktowego, co wiąże się z koniecznością napisania i wykonania dodatkowych testów).

Interesariusze są informowani o postępie w realizacji planu testów za pomocą raportów o postępie testów, które zawierają między innymi informacje o ewentualnych odchyleniach od planu oraz informacje pomagające uzasadnić podjęcie decyzji o wstrzymaniu testowania.

Kwestię monitorowania testów i nadzoru nad testami objaśniono dokładniej w podrozdziale 5.3.

Analiza testów

Grupa czynności „analiza testów” polega na przeanalizowanie podstawy testów w celu zidentyfikowania testowalnych cech i zdefiniowania związanych z nimi warunków testowych. Innymi słowy, analiza testów służy do ustalenia tego, „co” należy przetestować (w kategoriach mierzalnych kryteriów pokrycia).

Główne czynności wykonywane w ramach analizy testów to:

- dokonywanie analizy podstawy testów właściwej dla rozważanego poziomu testów, np.:
  - specyfikacji wymagań, takich jak: wymagania biznesowe, wymagania funkcjonalne, wymagania systemowe, historyjki użytkownika, opowieści (ang. *epic*)[2], przypadki użycia lub podobne produkty pracy, które określają pożądane zachowanie funkcjonalne i niefunkcjonalne modułu lub systemu;
  - informacji dotyczących projektu i implementacji, takich jak: grafy przepływu sterowania lub dokumenty opisujące architekturę systemu lub oprogramowania, specyfikacje projektowe, przepływy wywołań, modele oprogramowania (np. diagramy UML lub diagramy związków

[2] Autorzy tłumaczenia zdecydowali się przetłumaczyć angielskie pojęcie „*epic*” jako „opowieść”; spotykane jest też niepoprawne językowo tłumaczenie „epika”.

encji), specyfikacje interfejsów lub podobne produkty pracy, które określają strukturę modułu lub systemu;

- implementacji samego modułu lub systemu, w tym kodu, metadanych, zapytań do bazy danych oraz interfejsów;
- raportów z analizy ryzyka, które mogą dotyczyć aspektów funkcjonalnych, niefunkcjonalnych i strukturalnych modułu lub systemu;

- dokonywanie oceny testowalności podstawy testów i elementów testowych w celu zidentyfikowania często występujących typów defektów, które mogą powodować problemy z testowalnością, takich jak:
  - niejednoznaczności;
  - pominięcia;
  - niespójności;
  - nieścisłości;
  - sprzeczności;
  - nadmiarowe (zbędne) instrukcje;
- identyfikowanie cech i zbiorów cech, które mają zostać przetestowane;
- definiowanie warunków testowych w odniesieniu do poszczególnych cech oraz określenie ich priorytetów na podstawie analizy podstawy testów — z uwzględnieniem parametrów funkcjonalnych, niefunkcjonalnych i strukturalnych, innych czynników biznesowych i technicznych oraz poziomów ryzyka;
- stworzenie możliwości dwukierunkowego śledzenia powiązań między elementami podstawy testów a związanymi z nimi warunkami testowymi (patrz punkty 1.4.3. i 1.4.4.).

Zastosowanie technik czarnoskrzynkowych, białoskrzynkowych oraz technik opartych na doświadczeniu w ramach analizy testów (patrz rozdział 4.) pomaga zmniejszyć prawdopodobieństwo pominięcia ważnych warunków testowych oraz zdefiniować bardziej precyzyjne i dokładne warunki testowe.

W niektórych przypadkach w wyniku analizy testów definiowane są warunki testowe, które mają być wykorzystywane jako cele testów w kartach opisów testów. Karty opisu testów są typowymi produktami pracy w niektórych odmianach testowania opartego na doświadczeniu (patrz p. 4.4.2.). Jeśli istnieje śledzenie powiązań między wspomnianymi powyżej celami testów a podstawą testów, możliwy jest pomiar pokrycia uzyskanego w trakcie testowania opartego na doświadczeniu.

Identyfikowanie defektów na etapie analizy testów jest istotną potencjalną korzyścią, zwłaszcza gdy nie stosuje się żadnego innego procesu przeglądu i/lub gdy proces testowy jest ściśle powiązany z procesem przeglądu. Czynności wykonywane w ramach analizy testów pozwalają zweryfikować, czy wymagania są spójne, prawidłowo wyrażone i kompletne, a także dokonać ich walidacji, a więc sprawdzić, czy właściwie odzwierciedlają one potrzeby klienta, użytkowników i innych interesariuszy. Warto w tym miejscu podać przykład takich technik jak: wytwarzanie sterowane zachowaniem (ang. *Behavior Driven Development — BDD*) i wytwarzanie sterowane testami akceptacyjnymi (ang. *Acceptance Test Driven Development — ATDD*), które obejmuje generowanie warunków testowych i przypadków testowych na podstawie historyjek użytkownika i kryteriów akceptacji przed rozpoczęciem

tworzenia kodu. Technika ta przewiduje również weryfikację i walidację historyjek użytkownika i kryteriów akceptacji oraz wykrywanie występujących w nich defektów (patrz [ISTQB-CTFL- AT]).

Projektowanie testów

Podczas projektowania testów warunki testowe są przekształcane w przypadki testowe wysokiego poziomu, zbiory takich przypadków testowych oraz w inne testalia. W związku z tym, o ile analiza testów odpowiada na pytanie: „co należy przetestować?", o tyle projektowanie testów odpowiada na pytanie: „jak należy testować?".

Główne czynności wykonywane w ramach projektowania testów to:

- projektowanie przypadków testowych i zbiorów przypadków testowych oraz określenie ich priorytetów;
- identyfikowanie danych testowych niezbędnych do obsługi warunków testowych i przypadków testowych;
- projektowanie środowiska testowego oraz zidentyfikowanie wszelkich niezbędnych narzędzi i elementów infrastruktury;
- tworzenie możliwości dwukierunkowego śledzenia powiązań między podstawą testów, warunkami testowymi a przypadkami testowymi (patrz p. 1.4.4.).

Rozwinięcie warunków testowych w przypadki testowe i zbiory przypadków testowych na etapie projektowania testów wymaga często użycia odpowiednich technik testowania (patrz rozdział 4.).

Tak jak w przypadku analizy testów, projektowanie testów może również skutkować identyfikacją podobnych typów defektów w podstawie testów, co jest ważną potencjalną korzyścią.

Implementacja testów

Podczas implementacji testów tworzy się i/lub kończy testalia niezbędne do wykonania testów, w tym szeregowanie przypadków testowych w ramach procedur testowych. W związku z tym, o ile projektowanie testów odpowiada na pytanie: „jak testować?", o tyle implementacja testów odpowiada na pytanie: „czy mamy wszystko, co jest potrzebne do uruchomienia testów?".

Główne czynności wykonywane w ramach implementacji testów to:

- opracowanie procedur testowych i określenie ich priorytetów oraz, potencjalnie, utworzenie skryptów testów automatycznych;
- utworzenie zestawów testowych (na podstawie procedur testowych) oraz skryptów testów automatycznych (jeśli istnieją testy automatyczne);
- uporządkowanie zestawów testowych w harmonogram wykonywania testów w sposób zapewniający efektywny przebieg całego procesu (patrz p. 5.2.4.);
- zbudowanie środowiska testowego (włączając w to - jeśli to konieczne - jarzma testowe, wirtualizację usług, symulatory i inne elementy infrastrukturalne) oraz sprawdzenie, czy zostało ono poprawnie skonfigurowane;
- przygotowanie danych testowych i sprawdzenie, czy zostały poprawnie załadowane do środowiska testowego;

- zweryfikowanie i zaktualizowanie dwukierunkowego śledzenia powiązań między podstawą testów, warunkami testowymi, przypadkami testowymi, procedurami testowymi i zestawami testowymi (patrz p. 1.4.4.).

Zadania związane z projektowaniem i implementacją testów są często łączone.

Etapy projektowania i implementacji testów mogą być realizowane i dokumentowane w ramach wykonywania testów — zwłaszcza w przypadku testowania eksploracyjnego oraz innych typów testowania opartego na doświadczeniu. Testowanie eksploracyjne może odbywać się na podstawie kart opisu testów (sporządzanych w ramach analizy testów), przy czym testy eksploracyjne wykonywane są natychmiast po ich zaprojektowaniu i zaimplementowaniu (patrz p. 4.4.2.).

Wykonywanie testów

Podczas wykonywania testów uruchamiane są zestawy testowe, zgodnie z harmonogramem wykonywania testów. Główne czynności przeprowadzane w ramach wykonywania testów to:

- zarejestrowanie danych identyfikacyjnych i wersji elementów testowych bądź przedmiotu testów, narzędzi testowych i testaliów;
- wykonywanie testów ręcznie lub przy użyciu narzędzi do wykonywania testów;
- porównanie rzeczywistych wyników testów z wynikami oczekiwanymi;
- przeanalizowanie anomalii w celu ustalenia ich prawdopodobnych przyczyn (np. awarie mogą być wynikiem defektów w kodzie, ale mogą się pojawić również wyniki fałszywie pozytywne (patrz p. 1.2.3.));
- raportowanie defektów oparte na obserwowanych awariach (patrz podrozdział 5.6.);
- zarejestrowanie (zalogowanie) wyniku wykonania testów (np. zaliczenie, niezaliczenie, test blokujący);
- powtórzenie czynności testowych w wyniku działań podjętych w związku z wystąpieniem anomalii albo w ramach zaplanowanego testowania (np. testowania potwierdzającego, wykonywania poprawionego testu i/lub testowania regresji);
- zweryfikowanie i zaktualizowanie możliwości dwukierunkowego śledzenia powiązań między podstawą testów, warunkami testowymi, przypadkami testowymi, procedurami testowymi a wynikami testów.

Ukończenie testów

Ukończenie testów polega na zebraniu danych pochodzących z wykonanych czynności testowych w celu usystematyzowania i połączenia zdobytych doświadczeń, testaliów oraz innych istotnych informacji. Czynności związane z ukończeniem testów są wykonywane w momencie osiągnięcia kamieni milowych projektu, takich jak: przekazanie systemu lub oprogramowania do eksploatacji, zakończenie realizacji (lub anulowanie) projektu, zakończenie iteracji projektu zwinnego, ukończenie testów danego poziomu bądź zakończenie prac nad wydaniem pielęgnacyjnym (ang. *maintenance release*).

Główne czynności wykonywane w ramach ukończenia testów to:

- sprawdzenie, czy wszystkie raporty o defektach są zamknięte oraz wprowadzenie żądań zmian lub pozycji do rejestru produktu w odniesieniu do wszelkich defektów, które nie zostały rozwiązane do momentu zakończenia wykonywania testów;
- utworzenie sumarycznego raportu z testów, który zostanie przekazany interesariuszom;
- dla wersji końcowych: zarchiwizowanie środowiska testowego, danych testowych, infrastruktury testowej oraz innych testaliów, do ponownego wykorzystania w przyszłości;
- przekazanie testaliów zespołom odpowiedzialnym za pielęgnację, innym zespołom projektowym i/lub innym interesariuszom, którzy mogą odnieść korzyść z ich użycia;
- przeanalizowanie wniosków z ukończonych czynności testowych (ang. *lessons learned*) w celu ustalenia, jakie zmiany będą konieczne w przypadku przyszłych iteracji, wydań i projektów;
- wykorzystanie zebranych informacji do zwiększenia dojrzałości procesu testowego.

1.4.3. Produkty pracy związane z testowaniem

Produkty pracy związane z testowaniem powstają w ramach procesu testowego. Podobnie jak sposób implementacji procesu testowego, również typy produktów pracy powstających w wyniku tego procesu, sposób zarządzania tymi produktami i ich uporządkowania, a także nadawane im nazwy różnią się znacznie w poszczególnych organizacjach. W niniejszym sylabusie przyjęto opisaną powyżej definicję procesu testowego oraz produkty pracy (opisane zarówno w sylabusie, jak i w Słowniku terminów testowych ISTQB® [ISTQB S]). Punktem odniesienia dla produktów pracy związanych z testowaniem może być również standard międzynarodowy ISO/IEC/IEEE 29119-3.

Wiele z produktów pracy związanych z testowaniem, które opisano w tym punkcie, może być tworzonych i zarządzanych przy użyciu narzędzi do zarządzania testami oraz narzędzi do zarządzania defektami (patrz rozdział 6.).

Produkty pracy planowania testów

Wśród produktów pracy powstających na etapie planowania testów można zwykle wyróżnić jeden lub kilka planów testów. Plan testów zawiera informacje na temat podstawy testów, z którą zostaną powiązane za pośrednictwem informacji śledzenia inne produkty pracy związane z testowaniem (patrz poniżej i p. 1.4.4.). Ponadto określa się w nim kryteria wyjścia (definicję ukończenia), które będą stosowane w ramach monitorowania testów i nadzoru nad testami. Planowanie testów opisano w podrozdziale 5.2.

Produkty pracy monitorowania testów i nadzoru nad testami

Typowymi produktami pracy związanymi z monitorowaniem testów i nadzorem nad testami są różnego rodzaju raporty z testów, w tym raporty o postępie testów (tworzone na bieżąco i/lub w regularnych odstępach czasu) oraz sumaryczne raporty z testów (tworzone w momencie osiągnięcia poszczególnych kamieni milowych projektu). Raporty z testów powinny zawierać szczegółowe informacje na temat dotychczasowego przebiegu procesu testowego (w tym podsumowanie rezultatów wykonywania testów, gdy zostaną one udostępnione) i powinny być przedstawiane w formie dostosowanej do potrzeb konkretnych odbiorców. Monitorowanie testów i nadzór nad testami oraz produkty pracy powstające w ramach tych czynności opisano dokładniej w podrozdziale 5.3.

Produkty pracy monitorowania testów i nadzoru nad testami powinny również odnosić się do kwestii dotyczących zarządzania projektem, takich jak: ukończenie zadań, alokacja i zużycie zasobów czy też pracochłonność.

Produkty pracy analizy testów

Produkty pracy związane z analizą testów obejmują zdefiniowane i uszeregowane według priorytetów warunki testowe, przy czym pożądana jest możliwość dwukierunkowego śledzenia powiązań między tymi warunkami a pokrywanymi przez nie elementami podstawy testów. W przypadku testowania eksploracyjnego, w wyniku analizy testów mogą również powstawać karty opisu testów. Ponadto analiza testów może prowadzić do wykrycia i zgłoszenia defektów w podstawie testów.

Produkty pracy projektowania testów

W wyniku projektowania testów – w oparciu o zdefiniowane w analizie testów warunki testowe - powstają przypadki testowe i zbiory przypadków testowych.

Dobrą praktyką jest projektowanie przypadków testowych wysokiego poziomu, które nie zawierają konkretnych wartości danych wejściowych i oczekiwanych rezultatów. Wysokopoziomowe przypadki testowe można wykorzystywać wielokrotnie w różnych cyklach testowych z użyciem różnych konkretnych danych, należycie dokumentując przy tym zakres przypadku testowego. W idealnej sytuacji dla każdego przypadku testowego istnieje możliwość dwukierunkowego śledzenia powiązań między tym przypadkiem a pokrywanym przez niego warunkiem testowym (lub warunkami testowymi).

Wśród rezultatów etapu projektowania testów można również wymienić:

- zaprojektowanie i/lub zidentyfikowanie niezbędnych danych testowych;
- zaprojektowanie środowiska testowego;
- zidentyfikowanie infrastruktury i narzędzi,

przy czym stopień udokumentowania powyższych rezultatów bywa bardzo zróżnicowany.

Produkty pracy implementacji testów

Produkty pracy związane z implementacją testów to między innymi:

- procedury testowe oraz kolejność ich wykonywania;
- zestawy testowe;
- harmonogram wykonywania testów.

W idealnym przypadku po zakończeniu implementacji testów można wykazać spełnienie kryteriów pokrycia określonych w planie testów dzięki możliwości dwukierunkowego śledzenia powiązań między procedurami testowymi a elementami podstawy testów (za pośrednictwem przypadków testowych i warunków testowych).

W pewnych sytuacjach implementacja testów wiąże się również z utworzeniem produktów pracy, które wykorzystują narzędzia lub z których te narzędzia korzystają.. Przykłady takich produktów pracy to wirtualizacja usług czy skrypty testów automatycznych.

Implementacja testów może także skutkować utworzeniem i zweryfikowaniem danych testowych i środowiska testowego, przy czym poziom kompletności dokumentacji dotyczącej rezultatów weryfikacji danych i/lub środowiska może być bardzo różny.

Dane testowe służą do przypisywania konkretnych wartości do danych wejściowych i oczekiwanych rezultatów wykonania przypadków testowych. Te konkretne wartości, wraz z konkretnymi wskazówkami ich użycia, przekształcają przypadki testowe wysokiego poziomu w wykonywalne przypadki testowe niskiego poziomu. Ten sam przypadek testowy wysokiego poziomu można wykonywać z użyciem różnych danych testowych w odniesieniu do różnych wersji przedmiotu testów. Konkretne, oczekiwane rezultaty związane z określonymi danymi testowymi, identyfikuje się przy użyciu wyroczni testowej.

W przypadku testowania eksploracyjnego niektóre produkty pracy związane z projektowaniem i implementacją testów mogą powstawać dopiero w trakcie wykonywania testów, przy czym stopień udokumentowania testów eksploracyjnych i możliwości śledzenia powiązań z określonymi elementami podstawy testów może być bardzo różny.

Na etapie implementacji testów można również doprecyzować warunki testowe zdefiniowane podczas analizy testów.

Produkty pracy wykonywania testów

Do produktów pracy związanych z wykonywaniem testów zaliczają się między innymi:

- dokumentacja dotycząca statusu poszczególnych przypadków testowych lub procedur testowych (np.: gotowy do wykonania, zaliczony, niezaliczony, zablokowany, celowo pominięty itd.);
- raporty o defektach (patrz podrozdział 5.6.);
- dokumentacja wskazująca, które elementy testowe, przedmioty testów, narzędzia testowe i testalia zostały wykorzystane w ramach testowania.

W idealnym przypadku po zakończeniu wykonywania testów można ustalić status poszczególnych elementów podstawy testów i wygenerować raporty na ten temat dzięki możliwości dwukierunkowego śledzenia powiązań pomiędzy wspomnianymi powyżej elementami podstawy testów z odpowiednimi procedurami testowymi. Można na przykład wskazać, które wymagania zostały całkowicie przetestowane i pomyślnie przeszły testy, które wymagania nie przeszły testów i/lub wiążą się z defektami oraz które wymagania nie zostały jeszcze w pełni przetestowane. Umożliwia to sprawdzenie, czy zostały spełnione kryteria pokrycia testowego oraz przedstawienie rezultatów testów w raportach w sposób zrozumiały dla interesariuszy.

Produkty pracy ukończenia testów

Produkty pracy związane z ukończeniem testów to między innymi: sumaryczne raporty z testów, lista czynności do wykonania mających na celu wprowadzenie udoskonaleń w kolejnych projektach lub iteracjach, żądania zmian lub pozycje listy zaległości w backlogu produktu oraz ostateczna wersja testaliów.

1.4.4. Śledzenie powiązań między podstawą testów a produktami pracy związanymi z testowaniem

Jak wspomniano w p. 1.4.3., produkty pracy związane z testowaniem i ich nazwy mogą być bardzo różne. Niezależnie od tych różnic, w celu zapewnienia skutecznego monitorowania i nadzoru, istotne jest stworzenie i utrzymywanie mechanizmu śledzenia powiązań między każdym elementem podstawy testów a odpowiadającymi mu produktami pracy związanymi z testowaniem przez cały czas trwania procesu testowego. Sprawne śledzenie powiązań umożliwia nie tylko ocenę pokrycia testowego, ale również:

- analizowanie wpływu zmian;

- przeprowadzanie audytu testów;
- spełnianie kryteriów związanych z zarządzaniem w obszarze IT;
- tworzenie bardziej zrozumiałych raportów o statusie testów i sumarycznych raportów z testów dzięki uwzględnieniu statusu elementów podstawy testów (np. wymagań, dla których testy zostały zaliczone, niezaliczone lub czekają na wykonanie);
- przekazywanie interesariuszom informacji o aspektach technicznych testowania w zrozumiałej dla nich formie;
- udzielanie informacji potrzebnych do oceny jakości produktów, możliwości procesów i postępu w realizacji projektu z punktu widzenia możliwości osiągnięcia celów biznesowych.

Niektóre narzędzia do zarządzania testami mają wbudowane modele produktów pracy związanych z testowaniem, które odpowiadają pewnej części lub wszystkim produktom omówionym w tym punkcie. Ponadto istnieją organizacje, które budują własne systemy zarządzania w celu uporządkowania produktów pracy i dostarczania niezbędnych informacji związanych ze śledzeniem.

1.5. Psychologia testowania

Wytwarzanie oprogramowania, w tym jego testowanie, to proces realizowany z udziałem ludzi, w związku z czym duże znaczenie dla przebiegu testowania mają uwarunkowania psychologiczne.

1.5.1. Psychologia człowieka a testowanie

Identyfikowanie defektów podczas testowania statycznego (np. w ramach przeglądów wymagań lub sesji doprecyzowywania historyjek użytkowników) bądź identyfikowanie awarii w trakcie testowania dynamicznego może być odbierane jako krytyka produktu lub jego autora. Jednocześnie zjawisko psychologiczne zwane efektem potwierdzenia (ang. *confirmation bias*) może utrudniać zaakceptowanie informacji sprzecznych z dotychczasowymi przekonaniami. Przykładem takiego zjawiska są oczekiwania programistów, że ich kod będzie poprawny, co prowadzi do wystąpienia efektu potwierdzenia, który skutkuje tym, że trudno jest im zaakceptować fakt, że ich kod jest błędny. Obok efektu potwierdzenia mogą też występować inne błędy poznawcze, które utrudniają ludziom zrozumienie lub zaakceptowanie informacji uzyskanych w wyniku testowania. Dodatkowy czynnik utrudniający przyjęcie informacji zwrotnej z testów stanowi skłonność wielu osób do obwiniania osoby przynoszącej złe wiadomości, a takie sytuacje często są rezultatem testowania.

Na skutek działania powyższych czynników psychologicznych niektóre osoby mogą postrzegać testowanie jako czynność destrukcyjną, nawet jeśli przyczynia się ono wydatnie do postępu w realizacji projektu i podnoszenia jakości produktów (patrz podrozdziały 1.1. i 1.2.). Aby ograniczyć podobne reakcje, należy przekazywać informacje o defektach i awariach w sposób jak najbardziej konstruktywny, co pozwoli zmniejszyć napięcia między testerami a analitykami, właścicielami produktów, projektantami i programistami. Zasada ta ma zastosowanie zarówno do testowania statycznego, jak i do testowania dynamicznego.

Testerzy i kierownicy testów muszą posiadać duże umiejętności interpersonalne, aby sprawnie przekazywać informacje na temat defektów, awarii, rezultatów testów, postępu testowania i ryzyk oraz budować pozytywne relacje ze współpracownikami. Poniżej przedstawiono kilka zaleceń dotyczących zasad dobrej wymiany informacji:

- Należy zacząć od współpracy, a nie od konfliktu. Wszystkich powinna łączyć świadomość wspólnego celu, jakim jest podnoszenie jakości systemów.

- Należy podkreślić korzyści wynikające z testowania. Przykładem takiego zachowania jest wykorzystanie przez autorów informacji o defektach do doskonalenia produktów pracy i rozwijania umiejętności. Dla organizacji wykrycie i usunięcie defektów podczas testowania oznacza oszczędność czasu i pieniędzy oraz zmniejszenie ogólnego ryzyka związanego z jakością produktów.
- Informacje na temat rezultatów testów i inne wnioski należy przekazywać w sposób neutralny, koncentrując się na faktach i nie krytykując osoby, która stworzyła wadliwy element. W związku z powyższym należy zadbać o to, by raporty o defektach i wnioski z przeglądu były obiektywne oraz znajdowały oparcie w faktach.
- Należy wczuć się w sytuację drugiej osoby i zrozumieć, dlaczego negatywnie reaguje ona na podane informacje.
- Należy upewnić się, że rozmówca rozumie przekazywane informacje i *vice versa*.

Typowe cele testowania omówiono już wcześniej (patrz podrozdział 1.1.). Jednoznaczne zdefiniowanie właściwego zbioru celów testowania ma istotne implikacje psychologiczne, ponieważ większość osób ma skłonność do dopasowywania swoich planów i zachowań do celów określonych przez zespół, kierownictwo i innych interesariuszy. W związku z tym należy zadbać o to, by testerzy przestrzegali przyjętych założeń, a ich osobiste nastawienie w jak najmniejszym stopniu wpływało na wykonywaną pracę.

1.5.2. Różnice w sposobie myślenia testerów i programistów

Programiści i testerzy często prezentują odmienny sposób myślenia. Podstawowym celem prac programistycznych jest zaprojektowanie i wykonanie produktu. Jak wspomniano powyżej, cele testowania obejmują weryfikację i walidację produktu, wykrycie defektów przed przekazaniem produktu do eksploatacji itd. Tym samym mamy do czynienia z różnymi zbiorami celów wymagającymi różnego nastawienia, których pogodzenie jest kluczem do uzyskania produktu wyższej jakości.

Sposób myślenia danej osoby wyznaczają przyjmowane przez nią założenia oraz preferowane przez nią sposoby podejmowania decyzji i rozwiązywania problemów. Wśród pożądanych cech osobowościowych testera należy wymienić: ciekawość, „zawodowy pesymizm", umiejętność krytycznego spojrzenia na wykonywane czynności, dbałość o szczegóły oraz motywację do utrzymywania dobrej i pozytywnej komunikacji, a także relacji zawodowych. Warto też zaznaczyć, że sposób myślenia testera często ewoluuje i dojrzewa wraz ze zdobywaniem przez niego doświadczenia zawodowego.

Wśród typowych cech osobowościowych programisty mogą występować niektóre cechy charakterystyczne dla testera, ale programiści odnoszący sukcesy w swoim zawodzie są często bardziej zainteresowani projektowaniem i budowaniem rozwiązań niż analizowaniem ewentualnych problemów dotyczących przyjętych rozwiązań. Ważnym czynnikiem jest również efekt potwierdzenia, który może utrudnić uświadomienie sobie popełnionych przez siebie błędów.

Programiści mający odpowiednie nastawienie mogą testować własny kod. Różne modele cyklu życia oprogramowania oferują różne sposoby organizowania pracy testerów i wykonywania czynności testowych. Zlecając wykonanie niektórych czynności testowych niezależnym testerom, można zwiększyć skuteczność wykrywania defektów, co jest szczególnie ważne w przypadku systemów dużych, złożonych lub krytycznych ze względów bezpieczeństwa. Niezależni testerzy mają inny punkt widzenia, różny od punktów widzenia autorów produktów pracy (tj. analityków biznesowych, właścicieli produktów, projektantów i deweloperów), ponieważ mają inne uprzedzenia poznawcze (ang. *cognitive biases*).

2. Testowanie w cyklu życia oprogramowania — 100 min.

Słowa kluczowe

analiza wpływu, celów testu, oprogramowanie do powszechnej sprzedaży, podstawa testów, poziom testów, produkcyjne testy akceptacyjne, przedmiot testów, przypadek testowy, sekwencyjny model wytwarzania oprogramowania, środowisko testowe, testowanie akceptacyjne, testowanie akceptacyjne przez użytkownika, testowanie akceptacyjne zgodności z prawem, testowanie akceptacyjne zgodności z umową, testowanie alfa, testowanie beta, testowanie białoskrzynkowe, testowanie funkcjonalne, testowanie integracji modułów, testowanie integracji systemów, testowanie integracyjne, testowanie modułowe, testowanie niefunkcjonalne, testowanie pielęgnacyjne, testowanie potwierdzające, testowanie regresji, testowanie systemowe, testowanie związane ze zmianami, typ testów

Cele nauczania — testowanie w cyklu życia oprogramowania

2.1. Modele cyklu życia oprogramowania

FL-2.1.1. (K2) Kandydat potrafi wyjaśnić relacje między czynnościami związanymi z wytwarzaniem oprogramowania a czynnościami testowymi w cyklu życia oprogramowania.

FL-2.1.2. (K1) Kandydat potrafi wskazać powody, dla których konieczne jest dostosowanie modeli cyklu życia oprogramowania do kontekstu wynikającego z charakterystyki projektu i produktu.

2.2. Poziomy testów

FL-2.2.1. (K2) Kandydat potrafi porównać poszczególne poziomy testów z punktu widzenia celów, podstawy testów, przedmiotów testów, typowych defektów i awarii, podejść i odpowiedzialności.

2.3. Typy testów

FL-2.3.1. (K2) Kandydat potrafi porównać testowanie funkcjonalne z testowaniem niefunkcjonalnym i białoskrzynkowym.

FL-2.3.2. (K1) Kandydat zdaje sobie sprawę z tego, że testy funkcjonalne, niefunkcjonalne i białoskrzynkowe mogą występować na dowolnym poziomie testowania.

FL-2.3.3. (K2) Kandydat potrafi porównać przeznaczenie testowania potwierdzającego i testowania regresji.

2.4. Testowanie pielęgnacyjne

FL-2.4.1. (K2) Kandydat potrafi wymienić i omówić zdarzenia wyzwalające testowanie pielęgnacyjne.

FL-2.4.2. (K2) Kandydat potrafi opisać rolę analizy wpływu w testowaniu pielęgnacyjnym.

2.1. Modele cyklu życia oprogramowania

Model cyklu życia oprogramowania opisuje rodzaje czynności wykonywanych na poszczególnych etapach projektu wytwarzania oprogramowania oraz powiązania logiczne i chronologiczne między tymi czynnościami. Istnieje wiele różnych modeli cyklu życia oprogramowania, a każdy z nich wymaga innego podejścia do testowania.

2.1.1. Wytwarzanie oprogramowania a testowanie oprogramowania

Znajomość powszechnie stosowanych modeli cyklu życia oprogramowania jest ważnym elementem obowiązków każdego testera, ponieważ pozwala uwzględnić właściwe czynności testowe.

Poniżej wymieniono kilka zasad dobrych praktyk testowania, które dotyczą każdego modelu cyklu życia oprogramowania:

- dla każdej czynności związanej z wytwarzaniem oprogramowania istnieje odpowiadająca jej czynność testowa;
- każdy poziom testów ma przypisane cele odpowiednie do tego poziomu;
- analizę i projektowanie testów na potrzeby danego poziomu testów należy rozpocząć podczas wykonywania odpowiadającej danemu poziomowi czynności związanej z wytwarzaniem oprogramowania;
- testerzy powinni uczestniczyć w dyskusjach dotyczących definiowania i doprecyzowywania wymagań i założeń projektu oraz w przeglądach produktów pracy (np. wymagań, założeń projektu czy też historyjek użytkownika) natychmiast po udostępnieniu wersji roboczych odpowiednich dokumentów.

Bez względu na wybrany model cyklu życia oprogramowania czynności testowe powinny rozpocząć się w początkowym etapie tego cyklu zgodnie z zasadą wczesnego testowania.

W niniejszym sylabusie modele cyklu życia oprogramowania podzielono na następujące kategorie:

- sekwencyjne modele wytwarzania oprogramowania;
- iteracyjne i przyrostowe modele wytwarzania oprogramowania.

Sekwencyjny model wytwarzania oprogramowania opisuje proces wytwarzania oprogramowania jako liniowy przepływ czynności następujących kolejno po sobie. Oznacza to, że każda faza tego procesu powinna rozpoczynać się po zakończeniu fazy poprzedniej. W teorii poszczególne fazy nie zachodzą na siebie, ale w praktyce korzystne jest uzyskiwanie wczesnych informacji zwrotnych z kolejnej fazy.

W modelu kaskadowym (ang. *waterfall*) czynności związane z wytwarzaniem oprogramowania (takie jak: analiza wymagań, projektowanie, tworzenie kodu czy testowanie) wykonuje się jedna po drugiej. Zgodnie z założeniami tego modelu czynności testowe występują dopiero wtedy, gdy wszystkie inne czynności wytwórcze zostaną ukończone.

W przeciwieństwie do modelu kaskadowego model V zakłada integrację procesu testowania z całym procesem wytwarzania oprogramowania, a tym samym wprowadza w życie zasadę wczesnego testowania. Ponadto model V obejmuje poziomy testowania powiązane z poszczególnymi, odpowiadającymi im, fazami wytwarzania oprogramowania, co dodatkowo sprzyja wczesnemu testowaniu (patrz podrozdział 2.2.). W tym modelu wykonanie testów powiązanych ze wszystkimi poziomami testów następuje sekwencyjnie, jednak w niektórych przypadkach może następować nachodzenie faz na siebie nawzajem.

W większości przypadków sekwencyjne modele wytwarzania oprogramowania pozwalają na wytworzenie oprogramowania posiadającego pełny zestaw funkcjonalności. Stosowanie takich modeli wytwarzania oprogramowania zwykle wymaga miesięcy, jeśli nie lat, pracy, aby zostało ono dostarczone do interesariuszy i użytkowników.

Przyrostowe wytwarzanie oprogramowania to proces polegający na określaniu wymagań oraz projektowaniu, budowaniu i testowaniu systemu częściami, co oznacza, że funkcjonalność oprogramowania rośnie przyrostowo. Wielkość poszczególnych przyrostów funkcjonalności zależy od konkretnego modelu cyklu życia: niektóre modele przewidują podział na większe fragmenty, a inne — na mniejsze. Jednorazowy przyrost funkcjonalności może ograniczać się nawet do wprowadzenia pojedynczej zmiany na ekranie interfejsu użytkownika lub nowej opcji zapytania.

Wytwarzanie iteracyjne polega na specyfikowaniu, projektowaniu, budowaniu i testowaniu wspólnie grup funkcjonalności w serii cykli, często o określonym czasie trwania. Iteracje mogą zawierać zmiany funkcjonalności wytworzonych we wcześniejszych iteracjach, wspólnie ze zmianami w zakresie projektu. Każda iteracja dostarcza działające oprogramowanie, które stanowi rosnący podzbiór całkowitego zbioru funkcjonalności aż do momentu, w którym końcowa wersja oprogramowania zostanie dostarczona lub wytwarzanie zostanie zatrzymane.

Wśród przykładów modeli iteracyjnych można wyróżnić:

- *Rational Unified Process* (RUP): poszczególne iteracje trwają zwykle stosunkowo długo (np. dwa lub trzy miesiące), a przyrosty funkcjonalności są odpowiednio duże, czyli obejmują np. dwie lub trzy grupy powiązanych funkcjonalności.
- SCRUM: poszczególne iteracje trwają stosunkowo krótko (np. kilka godzin, dni lub tygodni), a przyrostowe części systemu są odpowiednio małe, czyli obejmują na przykład kilka udoskonaleń i dwie lub trzy nowe funkcjonalności.
- Kanban: wdrażany ze stałą lub zmienną długością iteracji umożliwia dostarczenie jednego udoskonalenia lub jednej funkcjonalności naraz (natychmiast po przygotowaniu) bądź zgrupowanie większej liczby funkcjonalności w celu równoczesnego przekazania na środowisko produkcyjne.
- Model spiralny: tworzy się eksperymentalne elementy przyrostowe, które następnie mogą zostać gruntownie przebudowane, a nawet porzucone na dalszych etapach wytwarzania oprogramowania.

Składowe komponenty systemów lub systemy wykorzystujące powyższe modele często odnoszą się do nachodzących na siebie i powtarzanych w cyklu życia oprogramowania poziomów testów. W idealnej sytuacji każda funkcjonalność jest testowana na wielu poziomach, zbliżając się do finalnego wydania. Czasami też zespoły korzystają z metody ciągłego dostarczania (ang. *Continuous Delivery*) lub ciągłego wdrażania oprogramowania. Te podejścia pozwalają na szerokie wykorzystywanie automatyzacji na wielu poziomach testowania (jako część potoku dostarczania oprogramowania). Ponadto wiele tego typu modeli uwzględnia koncepcję samoorganizujących się zespołów, która może zmienić sposób organizacji pracy w związku z testowaniem oraz relacje między testerami a programistami.

W każdym z powyższych modeli powstaje rosnący system, który może być dostarczany użytkownikom końcowym na zasadzie dodawania pojedynczych funkcjonalności, w określonych iteracjach lub w bardziej tradycyjny sposób, w formie dużych wydań. Niezależnie od tego, czy kolejne części przyrostowe oprogramowania są udostępniane użytkownikom, w miarę rozrastania się systemu coraz większego znaczenia nabiera testowanie regresji.

W przeciwieństwie do modeli sekwencyjnych, w modelach iteracyjno-przyrostowych działające oprogramowanie może być dostarczone już w ciągu tygodni, a nawet dni, jednak na spełnienie wszystkich wymagań potrzeba miesięcy, a nawet lat.

Więcej informacji na temat testowania oprogramowania w kontekście modeli wytwarzania zwinnego można znaleźć w [ISTQB CTFL-AT], a także w [Black 2017], [Crispin 2008], [Gregory 2015] oraz [Roman 2015].

2.1.2. Modele cyklu życia oprogramowania w kontekście

Modele cyklu życia oprogramowania należy dobierać i dopasowywać do kontekstu wynikającego z charakterystyki projektu i produktu. W związku z tym w procesie wyboru i dostosowania odpowiedniego do potrzeb projektu modelu należy wziąć pod uwagę: cel projektu, typ wytwarzanego produktu, priorytety biznesowe (takie jak czas wprowadzenia produktu na rynek) i zidentyfikowane czynniki ryzyka produktowego i projektowego. Przykładem może być wytwarzanie i testowanie wewnętrznego systemu administracyjnego o niewielkim znaczeniu, które powinno przebiegać inaczej niż wytwarzanie i testowanie systemu krytycznego ze względów bezpieczeństwa takiego jak układ sterowania hamulcami w samochodzie. Innym przykładem może być sytuacja, w której kwestie organizacyjne i kulturowe mogą utrudniać komunikację pomiędzy członkami zespołu, co może skutkować spowolnieniem wytwarzania oprogramowania w modelu iteracyjnym.

W zależności od kontekstu projektu, konieczne może okazać się połączenie lub reorganizacja niektórych poziomów testów i/lub czynności testowych. W przypadku integracji oprogramowania do powszechnej sprzedaży (ang: *Commercial off-the shelf* — COTS) z większym systemem nabywca może wykonywać testy współdziałania na poziomie testowania integracji systemów (np. w zakresie integracji z infrastrukturą i innymi systemami) oraz na poziomie testowania akceptacyjnego (testowanie funkcjonalne i niefunkcjonalne wraz z testowaniem akceptacyjnym przez użytkownika i produkcyjnymi testami akceptacyjnymi). Poziomy testów zostały dokładnie opisane w podrozdziale 2.2., a typy testów – w podrozdziale 2.3.

Różne modele cyklu życia oprogramowania można łączyć ze sobą. Przykładem może być zastosowanie modelu V do wytwarzania i testowania systemów back-endowych i ich integracji oraz modelu zwinnego do wytwarzania i testowania interfejsu użytkownika i funkcjonalności dostępnej dla użytkowników. Innym wariantem łączenia różnych modeli jest zastosowanie prototypowania na wczesnym etapie projektu, a następnie zastąpienie go modelem przyrostowym po zakończeniu fazy eksperymentalnej.

W przypadku systemów związanych z Internetem rzeczy (ang. *Internet of Things — IoT*), które składają się z wielu różnych obiektów takich jak: urządzenia, produkty i usługi, w odniesieniu do poszczególnych obiektów stosuje się zwykle oddzielne modele cyklu życia wytwarzania oprogramowania. Stanowi to duże wyzwanie — zwłaszcza w zakresie wytwarzania poszczególnych wersji systemów IoT. Ponadto w przypadku powyższych obiektów większy nacisk kładzie się na późniejsze etapy cyklu życia oprogramowania, już po wprowadzeniu obiektów na produkcję (np. na fazy: produkcyjną, aktualizacji i wycofania z użytku).

Poniżej podano powody, dla których modele rozwoju oprogramowania muszą być dostosowane do kontekstu projektu i charakterystyki produktu:

- różnica w ryzykach produktowych systemów (projekt złożony lub prosty);
- wiele jednostek biznesowych może być częścią projektu lub programu (połączenie rozwoju sekwencyjnego i zwinnego);
- krótki czas na dostarczenie produktu na rynek (łączenie poziomów testów i/lub integracja typów testów na poziomach testowych).

2.2. Poziomy testów

Poziomy testów to grupy czynności testowych, które organizuje się i którymi zarządza się wspólnie. Każdy poziom testów jest instancją procesu testowego składającą się z czynności opisanych w podrozdziale 1.4., wykonywanych dla oprogramowania na danym poziomie wytwarzania, od pojedynczych modułów lub komponentów, po kompletne systemy lub, jeśli ma to miejsce w danym przypadku, systemy systemów. Poziomy te są powiązane z innymi czynnościami wykonywanymi w ramach cyklu życia oprogramowania.

Poziomy testów opisane w niniejszym sylabusie to:

- testowanie modułowe;
- testowanie integracyjne;
- testowanie systemowe;
- testowanie akceptacyjne.

Każdy poziom testów charakteryzują następujące atrybuty:

- cele szczegółowe;
- podstawa testów (do której należy odwoływać się przy wyprowadzaniu przypadków testowych);
- przedmiot testów (czyli to, co jest testowane);
- typowe defekty i awarie;
- konkretne podejścia i odpowiedzialności.

Każdy poziom testów wymaga odpowiedniego środowiska testowego, np. do testowania akceptacyjnego idealnie nadaje się środowisko testowe podobne do środowiska produkcyjnego, a podczas testowania modułowego programiści zwykle korzystają z własnego środowiska rozwojowego.

2.2.1. Testowanie modułowe

Cele testowania modułowego

Testowanie modułowe (zwane także testowaniem jednostkowym, testowaniem komponentów lub testowaniem programu) skupia się na modułach, które można przetestować oddzielnie. Cele tego typu testowania to między innymi:

- zmniejszanie ryzyka;
- weryfikacja zgodności zachowań funkcjonalnych i niefunkcjonalnych modułu z projektem i specyfikacjami;
- budowanie zaufania do jakości modułu;
- wykrywanie defektów w module;
- zapobieganie przedostawaniu się defektów na wyższe poziomy testowania.

W niektórych przypadkach — zwłaszcza w przyrostowych i iteracyjnych modelach wytwarzania oprogramowania (np. modelu zwinnym), w których zmiany kodu mają charakter ciągły — automatyczne modułowe testy regresji są kluczowym elementem pozwalającym uzyskać pewność, że wprowadzone zmiany nie spowodowały nieprawidłowej pracy innych, istniejących i działających już modułów.

Testy modułowe są często wykonywane w izolacji od reszty systemu (zależnie od modelu cyklu życia oprogramowania i konkretnego systemu), przy czym w takiej sytuacji może być konieczne użycie atrap obiektów (ang. *mock object*), wirtualizacji usług, jarzm testowych, zaślepek bądź sterowników. Testowanie modułowe może obejmować funkcjonalność (np. poprawność obliczeń), parametry niefunkcjonalne (np. wycieki pamięci) oraz właściwości strukturalne (np. testowanie decyzji).

Podstawa testów

Przykładowe produkty pracy, które mogą być wykorzystywane jako podstawa testów w ramach testowania modułowego, to między innymi:

- projekt szczegółowy;
- kod;
- model danych;
- specyfikacje modułów.

Przedmioty testów

Do typowych przedmiotów testów dla testów modułowych zalicza się:

- moduły, jednostki lub komponenty;
- kod i struktury danych;
- klasy;
- moduły baz danych.

Typowe defekty i awarie

Przykładami typowych defektów i awarii wykrywanych w ramach testowania modułowego są:

- niepoprawna funkcjonalność (np. niezgodna ze specyfikacją projektu);
- problemy z przepływem danych;
- niepoprawny kod i logika.

Defekty są zwykle usuwane natychmiast po wykryciu, często bez formalnego procesu zarządzania nimi. Należy jednak zaznaczyć, że zgłaszając defekty, programiści dostarczają ważne informacje na potrzeby analizy przyczyny podstawowej i doskonalenia procesów.

Konkretne podejścia i odpowiedzialności

Testowanie modułowe jest zwykle wykonywane przez programistę będącego autorem kodu, a przynajmniej wymaga dostępu do testowanego kodu. W związku z tym programiści mogą naprzemiennie tworzyć moduły i wykrywać/usuwać defekty. Programiści często piszą i wykonują testy po napisaniu kodu danego modułu. Jednakże, w niektórych przypadkach — zwłaszcza w przypadku

zwinnego wytwarzania oprogramowania — automatyczne przypadki testowe do testowania modułowego można również tworzyć przed napisaniem kodu aplikacji.

Warto w tym miejscu omówić przykład wytwarzania sterowanego testami (ang. *Test Driven Development — TDD*). Model ten ma charakter wysoce iteracyjny i opiera się na cyklach obejmujących tworzenie automatycznych przypadków testowych, budowanie i integrowanie niewielkich fragmentów kodu, wykonywanie testów modułowych, rozwiązywanie ewentualnych problemów oraz refaktoryzację kodu. Proces ten jest powtarzany do momentu ukończenia modułu i zaliczenia wszystkich testów modułowych. Wytwarzanie sterowane testami jest przykładem podejścia typu „najpierw testuj". Model ten był pierwotnie stosowany w ramach programowania ekstremalnego (ang. *eXtreme Programming — XP*), ale upowszechnił się również w innych modelach zwinnych oraz cyklach sekwencyjnych (patrz [ISTQB CTFL-AT]).

2.2.2. Testowanie integracyjne

Cele testowania integracyjnego

Testowanie integracyjne skupia się na interakcjach między modułami lub systemami. Cele testowania integracyjnego to:

- zmniejszanie ryzyka;
- weryfikacja zgodności zachowań funkcjonalnych i niefunkcjonalnych z projektem i specyfikacjami;
- budowanie zaufania do jakości interfejsów;
- wykrywanie defektów (które mogą występować w samych interfejsach lub w modułach/systemach);
- zapobieganie przedostawaniu się defektów na wyższe poziomy testowania.

Podobnie jak w przypadku testowania modułowego istnieją sytuacje, w których automatyczne integracyjne testy regresji pozwalają uzyskać pewność, że wprowadzone zmiany nie spowodowały nieprawidłowej pracy dotychczas działających interfejsów, modułów lub systemów.

W niniejszym sylabusie opisano dwa różne poziomy testowania integracyjnego, na których testy mogą być wykonywane, w zależności od wielkości przedmiotów testów:

- Testowanie integracji modułów skupia się na interakcjach i interfejsach między integrowanymi modułami. Testy tego typu wykonuje się po zakończeniu testowania modułowego (zwykle są to testy automatyczne). W iteracyjnym i przyrostowym wytwarzaniu oprogramowania testy integracji modułów stanowią zwykle część procesu ciągłej integracji.
- Testowanie integracji systemów skupia się na interakcjach i interfejsach między systemami, pakietami i mikrousługami. Testy tego typu mogą również obejmować interakcje z interfejsami dostarczanymi przez organizacje zewnętrzne (np. dostawców usług internetowych — ang. *Web Services*). W takim przypadku organizacja wytwarzająca oprogramowanie nie kontroluje interfejsów zewnętrznych, co może stwarzać wiele problemów w obszarze testowania (związanych np. z usuwaniem defektów w kodzie tworzonym przez organizację zewnętrzną, które blokują testowanie, bądź z przygotowywaniem środowisk testowych). Testowanie integracji systemów może odbywać się po zakończeniu testowania systemowego lub równolegle do trwającego testowania systemowego (dotyczy to zarówno sekwencyjnego, jak i przyrostowego modelu wytwarzania oprogramowania).

Podstawa testów

Przykładowe produkty pracy, które mogą być wykorzystywane jako podstawa testów w ramach testowania integracyjnego, to między innymi:

- projekt oprogramowania i systemu;
- diagramy sekwencji;
- specyfikacje interfejsów i protokołów komunikacyjnych;
- przypadki użycia;
- architektura na poziomie modułów i systemu;
- przepływy pracy;
- definicje interfejsów zewnętrznych.

Przedmioty testów

Do typowych przedmiotów testów zalicza się:

- podsystemy;
- bazy danych;
- infrastrukturę;
- interfejsy;
- interfejsy programowania aplikacji (ang. *Application Programming Interface* — API);
- mikrousługi.

Typowe defekty i awarie

Przykładami typowych defektów i awarii wykrywanych w ramach testowania integracji modułów są:

- niepoprawne lub brakujące dane bądź niepoprawne kodowanie danych;
- niepoprawne uszeregowanie lub niepoprawna synchronizacja wywołań interfejsów;
- niezgodności interfejsów;
- błędy komunikacji między modułami;
- brak obsługi lub nieprawidłowa obsługa błędów komunikacji między modułami;
- niepoprawne założenia dotyczące znaczenia, jednostek lub granic danych przesyłanych między modułami.

Przykładami typowych defektów i awarii wykrywanych w ramach testowania integracji systemów są:

- niespójne struktury komunikatów przesyłanych między systemami;
- niepoprawne lub brakujące dane bądź niepoprawne kodowanie danych;
- niezgodność interfejsów;

- błędy komunikacji między systemami;
- brak obsługi lub nieprawidłowa obsługa błędów komunikacji między systemami;
- niepoprawne założenia dotyczące znaczenia, jednostek lub granic danych przesyłanych między systemami;
- nieprzestrzeganie rygorystycznych, obowiązujących przepisów dotyczących zabezpieczeń.

Konkretne podejścia i odpowiedzialności

Testowanie integracji modułów i testowanie integracji systemów powinno koncentrować się na samej integracji. Przykładem takiego podejścia jest integrowanie modułu A z modułem B, podczas którego testy powinny skupiać się na komunikacji między tymi modułami, a nie na funkcjonalności każdego z nich (funkcjonalności te powinny być przedmiotem testowania modułowego). Analogicznie w przypadku integrowania systemu X z systemem Y testerzy powinni skupiać się na komunikacji między tymi systemami, a nie na funkcjonalności każdego z nich (funkcjonalności te powinny być przedmiotem testowania systemowego). Na tym poziomie testów można przeprowadzać testy funkcjonalne, niefunkcjonalne i strukturalne.

Testowanie integracji modułów jest często obowiązkiem programistów, natomiast za testowanie integracji systemów zwykle odpowiadają testerzy. O ile jest to możliwe, testerzy wykonujący testowanie integracji systemów powinni znać architekturę systemu, a ich spostrzeżenia powinny być brane pod uwagę na etapie planowania integracji.

Zaplanowanie testów integracyjnych i strategii integracji przed zbudowaniem modułów lub systemów umożliwia wytworzenie tych modułów lub systemów w sposób zapewniający maksymalną efektywność testowania. Usystematyzowane strategie integracji mogą być oparte na architekturze systemu (np. strategie zstępujące lub wstępujące), zadaniach funkcjonalnych, sekwencjach przetwarzania transakcji bądź innych aspektach systemu lub modułów. Aby uprościć lokalizowanie defektów i umożliwić ich wczesne wykrywanie, integrację należy przeprowadzać metodą przyrostową (np. niewielka liczba jednocześnie dodawanych modułów lub systemów). Nie zaleca się integracji metodą „wielkiego wybuchu", polegającej na zintegrowaniu wszystkich modułów lub systemów naraz. W odpowiednim ukierunkowaniu testowania integracyjnego może również pomóc analiza ryzyka związanego z najbardziej złożonymi interfejsami.

Im szerszy jest zakres integracji, tym trudniej jest wskazać konkretny moduł lub system, w którym wystąpiły defekty, co może prowadzić do wzrostu ryzyka i wydłużenia czasu diagnozowania problemów. Jest to jeden z powodów, dla których powszechnie stosuje się metodę ciągłej integracji, polegającą na integrowaniu oprogramowania moduł po module (np. integracja funkcjonalna). Elementem ciągłej integracji jest często automatyczne testowanie regresji, które w miarę możliwości powinno odbywać się na wielu poziomach testów.

2.2.3. Testowanie systemowe

Cele testowania systemowego

Testowanie systemowe skupia się na zachowaniu i możliwościach całego systemu lub produktu, często z uwzględnieniem całokształtu zadań, jakie może on wykonywać oraz zachowań niefunkcjonalnych, jakie można stwierdzić podczas wykonywania tych zadań. Cele testowania systemowego to między innymi:

- zmniejszanie ryzyka;

- werfikacja zgodności zachowań funkcjonalnych i niefunkcjonalnych systemu z projektem i specyfikacjami;
- walidacja, czy system jest kompletny i działa zgodnie z oczekiwaniami;
- budowanie zaufania do jakości systemu jako całości;
- wykrywanie defektów;
- zapobieganie przedostawaniu się defektów na wyższe poziomy testowania lub na produkcję.

W przypadku niektórych systemów celem testowania może być również zweryfikowanie jakości danych. Podobnie jak w przypadku testowania modułowego i testowania integracyjnego, automatyczne, systemowe testy regresji pozwalają uzyskać pewność, że wprowadzone zmiany nie spowodowały nieprawidłowego działania dotychczasowych funkcji lub całościowej funkcjonalności. Często spotykanym rezultatem testowania systemowego są dane i informacje, na podstawie których interesariusze podejmują decyzje o przekazaniu systemu do użycia w produkcji. Ponadto testowanie systemowe może być niezbędne do spełnienia wymagań wynikających z obowiązujących przepisów prawa lub norm/standardów.

Środowisko testowe powinno w miarę możliwości odzwierciedlać specyfikę środowiska docelowego lub produkcyjnego.

Podstawa testów

Przykładowe produkty pracy, które mogą być wykorzystywane jako podstawa testów w ramach testowania systemowego, to między innymi:

- specyfikacje wymagań (funkcjonalnych i niefunkcjonalnych) dotyczących systemu i oprogramowania;
- raporty z analizy ryzyka;
- przypadki użycia;
- opowieści i historyjki użytkownika;
- modele zachowania systemu;
- diagramy stanów;
- instrukcje obsługi systemu i podręczniki użytkownika.

Przedmioty testów

Do typowych przedmiotów testów dla testów systemowych zalicza się:

- aplikacje;
- systemy łączące sprzęt i oprogramowanie;
- systemy operacyjne;
- system podlegający testowaniu;
- konfiguracja i dane konfiguracyjne systemu.

Typowe defekty i awarie

Przykładami typowych defektów i awarii wykrywanych w ramach testowania systemowego są:

- niepoprawne obliczenia;
- niepoprawne lub nieoczekiwane zachowania funkcjonalne lub niefunkcjonalne systemu;
- niepoprawne przepływy sterowania i/lub przepływy danych w systemie;
- problemy z prawidłowym i kompletnym wykonywaniem całościowych zadań funkcjonalnych;
- problemy z prawidłowym działaniem systemu w środowisku (środowiskach) testów systemowych;
- niezgodność działania systemu z opisami zawartymi w instrukcji obsługi systemu i podręcznikach użytkownika.

Konkretne podejścia i odpowiedzialności

Testowanie systemowe powinno koncentrować się na kompleksowym zbadaniu zachowania systemu jako całości na poziomie zarówno funkcjonalnym, jak i niefunkcjonalnym. W ramach testowania systemowego należy stosować techniki (patrz rozdział 4.), które są najbardziej odpowiednie dla danego aspektu (danych aspektów) systemu będącego przedmiotem testów. W celu sprawdzenia, czy zachowanie funkcjonalne jest zgodne z opisem zawartym w regułach biznesowych, można wykorzystać np. tablicę decyzyjną.

Testowanie systemowe jest zwykle przeprowadzane przez niezależnych testerów. W dużej mierze polegają oni na specyfikacjach. Defekty w specyfikacjach (np. brakujące historyjki użytkownika lub niepoprawnie zdefiniowane wymagania biznesowe) mogą doprowadzić do nieporozumień lub sporów dotyczących oczekiwanego zachowania systemu. Wyniki testów mogą więc zostać zaklasyfikowane jako rezultaty fałszywie pozytywne lub fałszywie negatywne, a sugerowanie się nimi może spowodować odpowiednio stratę czasu lub spadek skuteczności wykrywania defektów.

Między innymi dlatego (aby zmniejszyć częstość występowania powyższych sytuacji) należy zaangażować testerów w przeglądy, doprecyzowywanie historyjek użytkownika i inne czynności testowe o charakterze statycznym na jak najwcześniejszym etapie prac.

2.2.4. Testowanie akceptacyjne

Cele testowania akceptacyjnego

Testowanie akceptacyjne — podobnie jak testowanie systemowe — skupia się zwykle na zachowaniu i możliwościach całego systemu lub produktu. Cele testowania akceptacyjnego to najczęściej:

- budowanie zaufania do systemu;
- walidacja, czy system jest kompletny i czy będzie działał zgodnie z oczekiwaniami;
- weryfikacja zgodności zachowania funkcjonalnego i niefunkcjonalnego systemu ze specyfikacją.

W wyniku testowania akceptacyjnego mogą powstawać informacje pozwalające ocenić gotowość systemu do wdrożenia i użytkowania przez klienta (użytkownika). Podczas testowania akceptacyjnego mogą też zostać wykryte defekty, ale ich wykrywanie często nie jest celem tego poziomu testów (w niektórych przypadkach znalezienie dużej liczby defektów na etapie testowania akceptacyjnego może być nawet uznawane za istotny czynnik ryzyka projektowego). Ponadto testowanie akceptacyjne może być niezbędne do spełnienia wymagań wynikających z obowiązujących przepisów prawa lub norm/standardów.

Najczęściej występujące formy testowania akceptacyjnego to:

- testowanie akceptacyjne przez użytkownika;
- produkcyjne testy akceptacyjne;
- testowanie akceptacyjne zgodności z umową i zgodności z przepisami prawa;
- testowanie alfa i beta.

Formy testowania akceptacyjnego opisano w kolejnych czterech podpunktach.

Testowanie akceptacyjne przez użytkownika (ang. *User Acceptance Testing – UAT*)

Testowanie akceptacyjne systemu przez użytkownika odbywa się w (symulowanym) środowisku produkcyjnym i skupia się zwykle na walidacji, czy system nadaje się do użytkowania przez przyszłych odbiorców. Głównym celem jest tu zbudowanie pewności, że system umożliwi użytkownikom spełnienie ich potrzeb, postawionych przed nim wymagań i wykona proces biznesowy z minimalną liczbą problemów, minimalnymi kosztem i ryzykiem.

Produkcyjne testy akceptacyjne (ang. *Operational Acceptance Testing – OAT*)

Testowanie akceptacyjne systemu przez operatorów lub administratorów odbywa się zwykle w (symulowanym) środowisku produkcyjnym. Testy skupiają się na aspektach operacyjnych i mogą obejmować:

- testowanie mechanizmów tworzenia kopii zapasowych i odtwarzania danych;
- instalowanie, odinstalowywanie i aktualizowanie oprogramowania;
- usuwanie skutków awarii;
- zarządzanie użytkownikami;

- wykonywanie czynności pielęgnacyjnych;
- wykonywanie czynności związanych z ładowaniem i migracją danych;
- sprawdzanie, czy występują słabe punkty zabezpieczeń;
- wykonywanie testów wydajnościowych.

Głównym celem produkcyjnych testów akceptacyjnych jest uzyskanie pewności, że operatorzy lub administratorzy systemu będą w stanie zapewnić użytkownikom prawidłową pracę systemu w środowisku produkcyjnym, nawet w wyjątkowych i trudnych warunkach.

Testowanie akceptacyjne zgodności z umową i zgodności z przepisami prawa

Testowanie akceptacyjne zgodności z umową odbywa się zgodnie z kryteriami akceptacji zapisanymi w umowie dotyczącej wytworzenia oprogramowania na zlecenie. Powyższe kryteria akceptacji powinny zostać określone w momencie uzgadniania przez strony treści umowy. Tego rodzaju testowanie akceptacyjne wykonują często użytkownicy lub niezależni testerzy.

Testowanie akceptacyjne zgodności z przepisami prawa jest wykonywane w kontekście obowiązujących aktów prawnych, takich jak: ustawy, rozporządzenia czy normy bezpieczeństwa. Podobnie jak w przypadku testowania akceptacyjnego zgodności z umową, tego rodzaju testowanie akceptacyjne często wykonują użytkownicy lub niezależni testerzy, a rezultaty mogą być obserwowane lub kontrolowane przez organy nadzoru.

Głównym celem testowania akceptacyjnego zgodności z umową i zgodności z przepisami prawa jest uzyskanie pewności, że osiągnięto zgodność z wymaganiami wynikającymi z obowiązujących umów lub regulacji.

Testowanie alfa i beta

Twórcy oprogramowania przeznaczonego do powszechnej sprzedaży (COTS) często chcą uzyskać informacje zwrotne od potencjalnych lub obecnych klientów, zanim oprogramowanie trafi na rynek. Służą do tego testy alfa i beta.

Testowanie alfa jest wykonywane w siedzibie organizacji wytwarzającej oprogramowanie, ale zamiast zespołu wytwórczego testy wykonują potencjalni lub obecni klienci, i/lub operatorzy bądź niezależni testerzy. Z kolei testowanie beta wykonują obecni lub potencjalni klienci we własnych lokalizacjach. Testy beta mogą być, ale nie muszą, poprzedzone testami alfa.

Jednym z celów testów alfa i beta jest budowanie zaufania potencjalnych i aktualnych klientów i/lub operatorów w kwestii korzystania z systemu w normalnych warunkach, w środowisku produkcyjnym, aby osiągnąć swoje cele wkładając w to minimalny wysiłek, koszt i ryzyko. Innym celem tych testów może być wykrywanie błędów związanych z warunkami i środowiskiem (środowiskami), w których system będzie używany, szczególnie wtedy, gdy takie warunki są trudne do odtworzenia dla zespołu projektowego.

Podstawa testów

Przykładowe produkty pracy, które mogą być wykorzystywane jako podstawa testów w ramach dowolnego typu testowania akceptacyjnego, to między innymi:

- procesy biznesowe;
- wymagania użytkowników lub wymagania biznesowe;

- przepisy, umowy, normy i standardy;
- przypadki użycia i/lub historyjki użytkownika;
- wymagania systemowe;
- dokumentacja systemu lub podręczniki dla użytkowników;
- procedury instalacji;
- raporty z analizy ryzyka.

Ponadto podstawę testów, z której wyprowadzane są przypadki testowe na potrzeby produkcyjnych testów akceptacyjnych, mogą stanowić następujące produkty pracy:

- procedury tworzenia kopii zapasowych i odtwarzania danych;
- procedury usuwania skutków awarii;
- wymagania niefunkcjonalne;
- dokumentacja operacyjna;
- instrukcje wdrażania i instalacji;
- założenia wydajnościowe;
- pakiety bazodanowe;
- normy, standardy lub przepisy w dziedzinie zabezpieczeń.

Typowe przedmioty testów

Do typowych przedmiotów testów dowolnego typu testów akceptacyjnych zalicza się:

- system podlegający testowaniu;
- konfiguracja i dane konfiguracyjne systemu;
- procesy biznesowe wykonywane na całkowicie zintegrowanym systemie;
- systemy rezerwowe i ośrodki zastępcze (ang. *hot site*) służące do testowania mechanizmów zapewnienia ciągłości biznesowej i usuwania skutków awarii;
- procesy związane z użyciem produkcyjnym i utrzymaniem;
- formularze;
- raporty;
- istniejące i skonwertowane dane produkcyjne.

Typowe defekty i awarie

Przykładami typowych defektów wykrywanych w ramach różnych typów testowania akceptacyjnego są:

- systemowe przepływy pracy niezgodne z wymaganiami biznesowymi lub wymaganiami użytkowników;

- niepoprawnie zaimplementowane reguły biznesowe;
- niespełnienie przez system wymagań umownych lub prawnych;
- awarie niefunkcjonalne, takie jak słabe punkty zabezpieczeń, niedostateczna wydajność pod dużym obciążeniem bądź nieprawidłowe działanie na obsługiwanej platformie.

Konkretne podejścia i obowiązki

Testowanie akceptacyjne często spoczywa na klientach, użytkownikach biznesowych, właścicielach produktów lub operatorach systemów, ale w proces ten mogą być również zaangażowani inni interesariusze. Testowanie akceptacyjne zazwyczaj uznaje się za ostatni poziom sekwencyjnego modelu cyklu życia oprogramowania, ale może ono również odbywać się na innych jego etapach, np.:

- testowanie akceptacyjne oprogramowania „do powszechnej sprzedaży" może odbywać się podczas jego instalowania lub integrowania;
- testowanie akceptacyjne nowego udoskonalenia funkcjonalnego może mieć miejsce przed rozpoczęciem testowania systemowego.

W iteracyjnych modelach wytwarzania oprogramowania zespoły projektowe mogą stosować na zakończenie każdej iteracji różne formy testowania akceptacyjnego, takie jak testy ukierunkowane na weryfikację zgodności nowej funkcjonalności z kryteriami akceptacji bądź testy ukierunkowane na walidację nowej funkcjonalności z punktu widzenia potrzeb użytkowników. Ponadto pod koniec każdej iteracji, po ukończeniu każdej iteracji lub też po wykonaniu serii iteracji, mogą być wykonywane testy alfa i beta, a także testy akceptacyjne wykonywane przez użytkownika, produkcyjne testy akceptacyjne oraz testy akceptacyjne zgodności z umową i zgodności z przepisami prawa.

2.3. Typy testów

Typ testów to grupa dynamicznych czynności testowych wykonywanych z myślą o przetestowaniu określonych charakterystyk systemu/oprogramowania (lub jego części) zgodnie z określonymi celami testów. Celem testów może być między innymi:

- ocena funkcjonalnych charakterystyk jakościowych takich jak: kompletność, prawidłowość i adekwatność;
- dokonanie oceny niefunkcjonalnych charakterystyk jakościowych, w tym parametrów takich jak: niezawodność, wydajność, bezpieczeństwo, kompatybilność czy też użyteczność;
- ustalenie, czy struktura lub architektura modułu lub systemu jest poprawna, kompletna i zgodna ze specyfikacjami;
- dokonanie oceny skutków zmian, np. potwierdzenie usunięcia defektów (testowanie potwierdzające) lub wyszukanie niezamierzonych zmian w sposobie działania wynikających z modyfikacji oprogramowania lub środowiska (testowanie regresji).

2.3.1. Testowanie funkcjonalne

Testowanie funkcjonalne systemu polega na wykonaniu testów, które umożliwiają dokonanie oceny funkcji, jakie system ten powinien realizować. Wymagania funkcjonalne mogą być opisane w produktach pracy takich jak: specyfikacje wymagań biznesowych, opowieści, historyjki użytkownika, przypadki użycia lub specyfikacje funkcjonalne, ale zdarza się również, że występują one w postaci nieudokumentowanej. Funkcje opisują to, „co" powinien robić dany system.

Testy funkcjonalne należy wykonywać na wszystkich poziomach testów (np. testy dotyczące modułów mogą opierać się na specyfikacjach modułów), jednakże z zastrzeżeniem, że testy wykonywane na poszczególnych poziomach ukierunkowane są na różne zagadnienia (patrz podrozdział 2.2.).

Testowanie funkcjonalne uwzględnia zachowanie oprogramowania, w związku z czym do wyprowadzania warunków testowych i przypadków testowych dotyczących funkcjonalności modułu lub systemu można używać technik czarnoskrzynkowych (patrz podrozdział 4.2.).

Staranność testowania funkcjonalnego można zmierzyć na podstawie pokrycia funkcjonalnego. Termin „pokrycie funkcjonalne" oznacza stopień, w jakim została przetestowana funkcjonalność, wyrażony jako procent elementów danego typu pokrytych przez testy. Dzięki możliwości śledzenia powiązań między testami a wymaganiami funkcjonalnymi można na przykład obliczyć, jaki procent wymagań został uwzględniony w ramach testowania, a w rezultacie zidentyfikować ewentualne luki w pokryciu.

Do projektowania i wykonywania testów funkcjonalnych mogą być potrzebne specjalne umiejętności lub specjalna wiedza — taka jak znajomość konkretnego problemu biznesowego rozwiązywanego przy pomocy danego oprogramowana (np. oprogramowania do modelowania geologicznego dla przemysłu naftowego i gazowego) .

2.3.2. Testowanie niefunkcjonalne

Celem testowania niefunkcjonalnego jest dokonanie oceny charakterystyk systemów i oprogramowania, takich jak: użyteczność, wydajność, bezpieczeństwo itd. Klasyfikację charakterystyk jakościowych oprogramowania zawiera standard ISO/IEC 25010. Testowanie niefunkcjonalne pozwala sprawdzić to, „jak dobrze" zachowuje się dany system.

Wbrew mylnemu przekonaniu testowanie niefunkcjonalne można i często należy wykonywać na wszystkich poziomach testów. Ponadto powinno ono odbywać się na jak najwcześniejszym etapie, ponieważ zbyt późne wykrycie defektów niefunkcjonalnych może być bardzo dużym zagrożeniem dla powodzenia projektu.

Do wyprowadzania warunków testowych i przypadków testowych na potrzeby testowania niefunkcjonalnego można używać technik czarnoskrzynkowych (patrz podrozdział 4.2.). Przykładem może być zastosowanie analizy wartości brzegowych do zdefiniowania warunków skrajnych dotyczących testów wydajnościowych.

Staranność testowania niefunkcjonalnego można zmierzyć na podstawie pokrycia niefunkcjonalnego. Termin „pokrycie niefunkcjonalne" oznacza stopień, w jakim został przetestowany określony typ elementu niefunkcjonalnego, wyrażony jako procent elementów danego typu pokrytych przez testy. Dzięki możliwości śledzenia powiązań między testami a typami urządzeń obsługiwanych przez aplikację mobilną można na przykład obliczyć, jaki procent urządzeń został uwzględniony w ramach testowania kompatybilności, a w rezultacie zidentyfikować ewentualne luki w pokryciu.

Do projektowania i wykonywania testów niefunkcjonalnych mogą być potrzebne specjalne umiejętności lub specjalna wiedza — taka jak znajomość słabych punktów charakterystycznych dla danego projektu lub danej technologii (np. słabe punkty zabezpieczeń związanych z określonymi językami programowania) bądź konkretnej grupy użytkowników (np. profili użytkowników systemów do zarządzania placówkami opieki zdrowotnej).

Szczegółowe informacje na temat testowania niefunkcjonalnych charakterystyk jakościowych zawierają sylabusy [ISTQB-CTAL-TA], [ISTQB-CTAL-TTA], [ISTQB-CTAL-SEC] i inne specjalistyczne moduły ISTQB®.

2.3.3. Testowanie białoskrzynkowe

W przypadku testowania białoskrzynkowego warunki testowe wyprowadza się na podstawie struktury wewnętrznej lub implementacji danego systemu. Struktura wewnętrzna może obejmować kod, architekturę, przepływy pracy i/lub przepływy danych w obrębie systemu (patrz podrozdział 4.3.).

Staranność testowania białoskrzynkowego można zmierzyć na podstawie pokrycia strukturalnego. Termin „pokrycie strukturalne" oznacza stopień, w jakim został przetestowany określony typ elementu strukturalnego, wyrażony jako procent elementów danego typu pokrytych przez testy.

Na poziomie testowania modułów pokrycie kodu określa się na podstawie procentowej części kodu danego modułu, która została przetestowana. Wartość tę można mierzyć w kategoriach różnych aspektów kodu (przedmiotów pokrycia), takich jak procent instrukcji wykonywalnych lub procent wyników decyzji przetestowanych w danym module. Powyższe rodzaje pokrycia nazywa się zbiorczo pokryciem kodu. Na poziomie testowania integracji modułów, testowanie białoskrzynkowe może odbywać się na podstawie architektury systemu (np. interfejsów między modułami), a pokrycie strukturalne może być mierzone w kategoriach procentowego udziału przetestowanych interfejsów.

Do projektowania i wykonywania testów białoskrzynkowych mogą być potrzebne specjalne umiejętności lub specjalna wiedza, np.: znajomość struktury kodu, wiedza o sposobie przechowywania danych (pozwalająca np. ocenić zapytania do baz danych) bądź sposób korzystania z narzędzi do pomiaru pokrycia i poprawnego interpretowania generowanych przez nie rezultatów.

2.3.4. Testowanie związane ze zmianami

Po wprowadzeniu w systemie zmian mających na celu usunięcie defektów bądź dodanie lub zmodyfikowanie funkcjonalności należy przeprowadzić testy, które potwierdzą, że wprowadzone zmiany faktycznie skutkowały naprawieniem defektu lub poprawną implementacją odpowiedniej funkcjonalności, a przy tym nie wywołały żadnych nieprzewidzianych, niekorzystnych zachowań oprogramowania.

- Testowanie potwierdzające. Po naprawieniu defektu można przetestować oprogramowanie przy użyciu wszystkich przypadków testowych, które wcześniej nie zostały zaliczone z powodu wystąpienia tego defektu, a powinny zostać ponownie wykonane w nowej wersji oprogramowania. Oprogramowanie może być również testowane przy pomocy nowych testów, by pokryć zmiany niezbędne do usunięcia usterki. Absolutnym minimum jest ponowne wykonanie w nowej wersji oprogramowania kroków niezbędnych do odtworzenia awarii wywołanej przez dany defekt. Celem testu potwierdzającego jest sprawdzenie, czy pierwotny defekt został pomyślnie usunięty.

- Testowanie regresji. Istnieje ryzyko, że zmiana (poprawka lub inna modyfikacja) wprowadzona w jednej części kodu wpłynie przypadkowo na zachowanie innych części kodu w tym samym module, w innych modułach tego samego systemu, a nawet w innych systemach. Ponadto należy wziąć pod uwagę zmiany dotyczące środowiska, takie jak wprowadzenie nowej wersji systemu operacyjnego lub systemu zarządzania bazami danych. Powyższe, niezamierzone skutki uboczne, są nazywane regresjami, a testowanie regresji polega na ponownym wykonaniu testów w celu ich wykrycia.

Testowanie potwierdzające i testowanie regresji można wykonywać na wszystkich poziomach testów.

Zwłaszcza w iteracyjnych i przyrostowych cyklach życia oprogramowania (np. w podejściu zwinnym) nowe funkcjonalności, zmiany dotychczasowych funkcjonalności oraz refaktoryzacja powodują częste zmiany kodu, które pociągają za sobą konieczność przeprowadzenia odpowiednich testów. Z uwagi na ciągłą ewolucję systemu, testowanie potwierdzające i testowanie regresji są bardzo istotne, szczególnie

w przypadku systemów związanych z Internetem rzeczy (IoT), w których poszczególne obiekty (np. urządzenia) są często aktualizowane lub wymieniane.

Zestawy testów regresji są wykonywane wielokrotnie i zwykle ewoluują dość wolno, w związku z czym testowanie regresji świetnie nadaje się do automatyzacji - dlatego też automatyzację tego rodzaju testów należy rozpocząć w początkowym etapie projektu (patrz rozdział 6.).

2.3.5. Poziomy testów a typy testów

Każdy z wymienionych powyżej typów testów można wykonywać na dowolnym poziomie testów. Aby zilustrować tę prawidłowość, poniżej przedstawiono przykłady testów funkcjonalnych, niefunkcjonalnych i białoskrzynkowych oraz testów związanych ze zmianami, które są wykonywane na wszystkich poziomach testów w odniesieniu do aplikacji bankowej. Poniżej podano przykłady testów funkcjonalnych wykonywanych na różnych poziomach testów:

- na potrzeby testowania modułowego projektowane są testy odzwierciedlające sposób, w jaki dany moduł powinien obliczać odsetki składane;
- na potrzeby testowania integracji modułów projektowane są testy odzwierciedlające sposób, w jaki informacje na temat konta pozyskane poprzez interfejs użytkownika są przekazywane do warstwy logiki biznesowej;
- na potrzeby testowania systemowego projektowane są testy odzwierciedlające sposób, w jaki posiadacze rachunków mogą składać wnioski o przyznanie linii kredytowej na rachunku bieżącym;
- na potrzeby testowania integracji systemów projektowane są testy odzwierciedlające sposób, w jaki dany system sprawdza zdolność kredytową posiadacza rachunku przy użyciu zewnętrznej mikrousługi;
- na potrzeby testowania akceptacyjnego projektowane są testy odzwierciedlające sposób, w jaki pracownik banku zatwierdza lub odrzuca wniosek kredytowy.

Poniżej opisano przykłady testów niefunkcjonalnych przeprowadzanych na różnych poziomach testów:

- na potrzeby testowania modułowego projektowane są testy wydajnościowe, które umożliwiają ocenę liczby cykli procesora niezbędnych do wykonania złożonych obliczeń dotyczących łącznej kwoty odsetek;
- na potrzeby testowania integracji modułów projektowane są testy zabezpieczeń, które mają na celu wykrycie słabych punktów zabezpieczeń związanych z przepełnieniem bufora danymi przekazywanymi z interfejsu użytkownika do warstwy logiki biznesowej;
- na potrzeby testowania systemowego projektowane są testy przenaszalności, które umożliwiają sprawdzenie, czy warstwa prezentacji działa we wszystkich obsługiwanych przeglądarkach i urządzeniach przenośnych;
- na potrzeby testowania integracji systemów projektowane są testy niezawodności, które pozwalają na dokonanie oceny odporności systemu w przypadku braku odpowiedzi ze strony mikrousługi służącej do sprawdzania zdolności kredytowej;
- na potrzeby testowania akceptacyjnego projektowane są testy użyteczności, które pozwalają ocenić ułatwienia dostępu dla osób niepełnosprawnych zastosowane w interfejsie do przetwarzania kredytów po stronie banku.

Kolejna grupa zawiera przykłady testów białoskrzynkowych przeprowadzanych na różnych poziomach testów:

- na potrzeby testowania modułowego projektowane są testy, których celem jest zapewnienie pełnego pokrycia instrukcji kodu i decyzji (patrz podrozdział 4.3.) we wszystkich modułach wykonujących obliczenia finansowe;
- na potrzeby testowania integracji modułów projektowane są testy, które umożliwiają sprawdzenie, w jaki sposób każdy ekran interfejsu wyświetlanego w przeglądarce przekazuje dane prezentowane na następnym ekranie oraz do warstwy logiki biznesowej;
- na potrzeby testowania systemowego projektowane są testy, których celem jest zapewnienie pokrycia możliwych sekwencji stron internetowych wyświetlanych podczas składania wniosku o przyznanie linii kredytowej;
- na potrzeby testowania integracji systemów projektowane są testy, których celem jest sprawdzenie wszystkich możliwych typów zapytań wysyłanych do mikrousługi służącej do sprawdzania zdolności kredytowej;
- na potrzeby testowania akceptacyjnego projektowane są testy, których celem jest pokrycie wszystkich obsługiwanych struktur i zakresów wartości dla plików z danymi finansowymi używanych w przelewach międzybankowych.

Ostatnia grupa zawiera przykłady testów związanych ze zmianami przeprowadzanych na różnych poziomach testów:

- na potrzeby testowania modułowego projektowane są automatyczne testy regresji dotyczące poszczególnych modułów (testy te zostaną następnie uwzględnione w strukturze ciągłej integracji);
- na potrzeby testowania integracji modułów projektowane są testy służące do potwierdzania skuteczności wprowadzonych poprawek związanych z defektami w interfejsach w miarę umieszczania takich poprawek w repozytorium kodu;
- na potrzeby testowania systemowego ponownie wykonywane są wszystkie testy dotyczące danego przepływu pracy, jeśli dane czy sposób ich prezentacji na którymkolwiek z ekranów objętych tym przepływem pracy uległ zmianie;
- na potrzeby testowania integracji systemów codziennie wykonywane są ponownie testy aplikacji współpracującej z mikrousługą do sprawdzania zdolności kredytowej (w ramach ciągłego wdrażania tej mikrousługi);
- na potrzeby testowania akceptacyjnego wszystkie wcześniej niezaliczone testy są wykonywane ponownie (po usunięciu defektu wykrytego w ramach poprzedniego testowania akceptacyjnego).

Powyżej przedstawiono przykłady wszystkich typów testów wykonywanych na wszystkich poziomach testów, jednak nie każde oprogramowanie wymaga uwzględnienia każdego typu testów na każdym poziomie. Ważne jest to, aby na poszczególnych poziomach zostały wykonane odpowiednie testy — dotyczy to zwłaszcza pierwszego z poziomów testów, na jakim występuje dany typ testów.

2.4. Testowanie pielęgnacyjne

Po wdrożeniu w środowisku produkcyjnym oprogramowanie lub system wymaga dalszej pielęgnacji. Różnego rodzaju zmiany — związane np. z usuwaniem defektów uwidocznionych przez awarie

zaistniałe podczas użytkowania produkcyjnego, dodaniem nowej funkcjonalności bądź usuwaniem lub modyfikowaniem funkcjonalności już istniejącej — są praktycznie nieuniknione. Ponadto pielęgnacja jest niezbędna do utrzymania lub poprawy wymaganych niefunkcjonalnych charakterystyk jakościowych oprogramowania lub systemu przez cały cykl jego życia — zwłaszcza w zakresie takich parametrów jak: wydajność, kompatybilność, niezawodność, bezpieczeństwo i przenaszalność.

Po dokonaniu każdej zmiany w fazie pielęgnacji, należy wykonać testowanie pielęgnacyjne, którego celem jest zarówno sprawdzenie, czy zmiana została wprowadzona pomyślnie, jak i wykrycie ewentualnych, niezamierzonych skutków ubocznych (np. regresji) w niezmienionych częściach systemu (czyli zwykle w większości jego obszarów). Pielęgnacja może być wykonywana zarówno planowo (w związku z nowymi wersjami), jak i w sposób niezaplanowany (w związku z poprawkami doraźnymi — ang. *hotfix*).

Wydanie pielęgnacyjne może wymagać wykonania testów pielęgnacyjnych na wielu poziomach testów i z wykorzystaniem różnych typów testów — zależnie od zakresu wprowadzonych zmian. Na zakres testowania pielęgnacyjnego wpływają między innymi:

- poziom ryzyka związanego ze zmianą (np. stopień, w jakim zmieniony obszar oprogramowania komunikuje się z innymi modułami lub systemami);
- wielkość dotychczasowego systemu;
- wielkość wprowadzonej zmiany.

2.4.1. Zdarzenia wywołujące pielęgnację

Istnieje kilka powodów, dla których wykonuje się pielęgnację oprogramowania, a tym samym testowanie pielęgnacyjne. Dotyczy to zarówno zmian planowanych, jak i nieplanowanych.

Zdarzenia wywołujące pielęgnację można podzielić na następujące kategorie:

- Modyfikacja. Ta kategoria obejmuje między innymi: zaplanowane udoskonalenia (np. w postaci nowych wersji oprogramowania), zmiany korekcyjne i awaryjne, zmiany środowiska operacyjnego (np. planowe uaktualnienia systemu operacyjnego lub bazy danych), uaktualnienia oprogramowania do powszechnej sprzedaży (COTS) oraz poprawki usuwające defekty i słabe punkty zabezpieczeń.
- Migracja. Ta kategoria obejmuje między innymi przejście z jednej platformy na inną, co może wiązać się z koniecznością przeprowadzenia testów produkcyjnych nowego środowiska i zmienionego oprogramowania bądź testów konwersji danych (w przypadku migracji danych z innej aplikacji do pielęgnowanego systemu).
- Wycofanie. Ta kategoria dotyczy sytuacji, w której użycie aplikacji dobiega końca. Kiedy aplikacja lub system jest wycofywany, może to wymagać testowania migracji lub archiwizacji danych, jeśli zachodzi potrzeba ich przechowywania przez dłuższy czas;
    - testowanie procedur odzyskiwania/pozyskiwania zarchiwizowanych danych przez dłuższy czas może także być konieczne:
    - nieodzowne może być uwzględnienie testów regresji, aby zapewnić dalszą prawidłową pracę funkcjonalności pozostających w użyciu.

W przypadku systemów związanych z Internetem rzeczy (IoT – *Internet of Things*) testowanie pielęgnacyjne może być konieczne po wprowadzeniu w systemie zupełnie nowych lub zmodyfikowanych elementów, takich jak urządzenia sprzętowe czy usługi programowe. Podczas

testowania pielęgnacyjnego tego typu systemów szczególny nacisk kładzie się na testowanie integracyjne na różnych poziomach (np. na poziomie sieci i aplikacji) oraz na aspekty związane z zabezpieczeniami, szczególnie w zakresie danych osobowych.

2.4.2. Analiza wpływu związana z pielęgnacją

Analiza wpływu pozwala ocenić zmiany wprowadzone w wersji pielęgnacyjnej pod kątem zarówno skutków zamierzonych, jak i spodziewanych lub potencjalnych skutków ubocznych, a także umożliwia zidentyfikowanie obszarów systemu, na które będą miały wpływ wprowadzone zmiany. Ponadto może pomóc w zidentyfikowaniu wpływu zmiany na dotychczasowe testy. Skutki uboczne zmiany oraz obszary systemu, na które może ona wpływać, należy przetestować pod kątem regresji, przy czym czynność ta może być poprzedzona aktualizacją istniejących testów, na które oddziałuje dana zmiana.

Analizę wpływu można przeprowadzić przed dokonaniem zmiany, aby ustalić, czy zmianę tę należy faktycznie wprowadzić (z uwagi na potencjalne konsekwencje dla innych obszarów systemu).

Przeprowadzenie analizy wpływu może być utrudnione, jeśli:

- specyfikacje (np. wymagania biznesowe, historyjki użytkownika, architektura) są nieaktualne lub niedostępne;
- przypadki testowe nie zostały udokumentowane lub są nieaktualne;
- nie stworzono możliwości dwukierunkowego śledzenia powiązań między testami a podstawą testów;
- wsparcie narzędziowe nie istnieje lub jest niewystarczające;
- zaangażowane osoby nie dysponują wiedzą z danej dziedziny i/lub na temat danego systemu;
- podczas wytwarzania oprogramowania poświęcono zbyt mało uwagi jego charakterystyce jakościowej w zakresie utrzymywalności.

3. Testowanie statyczne — 135 min.

Słowa kluczowe

analiza statyczna, czytanie oparte na perspektywie, inspekcja, przegląd, przegląd *ad hoc*, przegląd formalny, przegląd nieformalny, przegląd oparty na liście kontrolnej, przegląd oparty na rolach, przegląd oparty na scenariuszach, przegląd techniczny, przejrzenie, testowanie dynamiczne, testowanie statyczne

Cele nauczania — testowanie statyczne

3.1. Podstawy testowania statycznego

FL-3.1.1. (K1) Kandydat potrafi rozpoznać typy produktów pracy związanych z oprogramowaniem, które mogą być badane przy użyciu poszczególnych technik testowania statycznego.

FL-3.1.2. (K2) Kandydat potrafi opisać podając przykłady wartość testowania statycznego.

FL-3.1.3. (K2) Kandydat potrafi wyjaśnić różnicę między technikami testowania statycznego a technikami testowania dynamicznego z uwzględnieniem celów, typów identyfikowanych defektów oraz roli tych technik w cyklu życia oprogramowania.

3.2. Proces przeglądu

FL-3.2.1. (K2) Kandydat potrafi podsumować czynności wykonywane w ramach procesu przeglądu produktów pracy.

FL-3.2.2. (K1) Kandydat potrafi rozpoznać poszczególne role i obowiązki w przeglądzie formalnym.

FL-3.2.3. (K2) Kandydat potrafi wyjaśnić różnice między poszczególnymi typami przeglądów: przeglądem nieformalnym, przejrzeniem, przeglądem technicznym i inspekcją.

FL-3.2.4. (K3) Kandydat potrafi zastosować technikę przeglądu do produktu pracy w celu wykrycia defektów.

FL-3.2.5. (K2) Kandydat potrafi wyjaśnić, jakie czynniki decydują o powodzeniu przeglądu.

3.1. Podstawy testowania statycznego

W przeciwieństwie do testowania dynamicznego, które wymaga uruchomienia testowanego oprogramowania, testowanie statyczne opiera się na ręcznym badaniu produktów pracy (tj. wykonywaniu przeglądów) bądź dokonywaniu oceny kodu przy użyciu odpowiednich narzędzi lub innych produktów pracy (tj. analizie statycznej). Oba typy testowania statycznego pozwalają ocenić testowany kod lub inny produkt pracy bez jego uruchamiania.

Analiza statyczna jest szczególnie ważna w przypadku systemów komputerowych krytycznych ze względów bezpieczeństwa (np. systemów stosowanych w lotnictwie, medycynie lub w energetyce jądrowej), ale staje się też coraz bardziej istotna i powszechnie stosowana w innych kontekstach (analiza taka jest np. ważnym elementem testowania zabezpieczeń). Ponadto analiza statyczna jest często elementem automatycznych narzędzi do budowy i dystrybucji systemów, które to narzędzia są stosowane np. w podejściu zwinnym, ciągłym dostarczaniu i ciągłym wdrażaniu oprogramowania.

3.1.1. Produkty pracy badane metodą testowania statycznego

Przy użyciu technik testowania statycznego (tj. przeglądów i/lub analizy statycznej) można zbadać niemal wszystkie produkty pracy, na przykład:

- specyfikacje (w tym: wymagania biznesowe, wymagania funkcjonalne i wymagania w zakresie bezpieczeństwa);
- opowieści, historyjki użytkownika i kryteria akceptacji;
- architekturę i specyfikacje projektowe;
- kod;
- testalia (w tym: plany testów, przypadki testowe, procedury testowe, i skrypty testów automatycznych);
- podręczniki użytkownika;
- strony internetowe;
- umowy, plany projektów, harmonogramy i plany budżetowe;
- konfigurację i infrastrukturę,
- modele takie jak diagramy aktywności, które mogą być używane do testowania opartego na modelu (patrz [ISTQB-CTFL-MBT] i [Kramer 2016]).

Przeglądy mogą dotyczyć dowolnego produktu pracy, który uczestnicy potrafią przeczytać i zrozumieć. Z kolei analizę statyczną można zastosować efektywnie do każdego produktu pracy o strukturze formalnej (w typowych sytuacjach to kod lub modele), dla którego istnieje odpowiednie narzędzie do jej przeprowadzenia. Dostępne są również narzędzia pozwalające oceniać produkty pracy napisane w języku naturalnym takie jak wymagania (np. sprawdzające te produkty pracy pod kątem pisowni, gramatyki czy też czytelności).

3.1.2. Zalety testowania statycznego

Techniki testowania statycznego dostarczają wielu korzyści. W przypadku ich stosowania w początkowym etapie cyklu życia oprogramowania, pozwalają one wykryć defekty jeszcze przed rozpoczęciem testowania dynamicznego (np. w ramach przeglądów specyfikacji wymagań lub projektu bądź w backlogu produktu itp.). Usunięcie defektów wykrytych w początkowym etapie cyklu życia

oprogramowania jest często dużo tańsze niż usunięcie defektów wykrytych w kolejnych etapach — zwłaszcza po wdrożeniu oprogramowania i rozpoczęciu jego eksploatacji.

Zastosowanie technik testowania statycznego do wykrywania defektów i ich niezwłocznego usuwania jest prawie zawsze dużo tańsze niż wykrywanie defektów w przedmiocie testów i ich usuwanie metodą testowania dynamicznego, szczególnie jeżeli weźmie się pod uwagę dodatkowe koszty związane z aktualizacją pozostałych produktów pracy oraz testowaniem potwierdzającym i testowaniem regresji.

Wśród dodatkowych zalet testowania statycznego wymienić można:

- efektywne wykrywanie i usuwanie defektów jeszcze przed wykonaniem testów dynamicznych;
- identyfikowanie defektów, które trudno jest wykryć metodą testowania dynamicznego;
- zapobieganie wystąpieniu defektów w projekcie i kodzie poprzez wykrywanie niedociągnięć takich jak: niespójności, niejednoznaczności, wewnętrzne sprzeczności, przeoczenia, opuszczenia, nieścisłości czy też elementy nadmiarowe w wymaganiach;
- zwiększenie wydajności prac programistycznych między innymi dzięki udoskonaleniom projektowania i utrzymywalności kodu;
- obniżenie kosztów i zmniejszenie czasochłonności wytwarzania oprogramowania;
- obniżenie kosztów i zmniejszenie czasochłonności testowania;
- obniżenie łącznego kosztu zapewnienia jakości w całym cyklu życia oprogramowania poprzez zmniejszenie liczby awarii na późniejszych etapach tego cyklu lub po przekazaniu oprogramowania do eksploatacji;
- usprawnienie komunikacji pomiędzy członkami zespołu uczestniczącymi w przeglądach.

3.1.3. Różnice między testowaniem statycznym a dynamicznym

Testowanie statyczne i testowanie dynamiczne mogą mieć te same cele (patrz p. 1.1.1.), takie jak dokonywanie oceny jakości produktów pracy czy jak najwcześniejsze identyfikowanie defektów. Czynności te uzupełniają się wzajemnie, ponieważ umożliwiają wykrywanie różnych typów defektów.

Główna różnica pomiędzy testowaniem statycznym a dynamicznych polega na tym, że testowanie statyczne pozwala wykryć defekty bezpośrednio w produktach pracy, a nie na podstawie spowodowanych przez te defekty awarii zidentyfikowanych podczas uruchamiania oprogramowania. Zdarza się, że defekt w produkcie pracy pozostaje przez bardzo długi czas ukryty, ponieważ nie powoduje awarii. Ścieżka, na której się on znajduje, może być rzadko testowana lub trudno dostępna, przez co nie będzie łatwo zaprojektować i wykonać test dynamiczny, który go wykryje. Testowanie statyczne może umożliwić znalezienie defektu przy znacznie mniejszym nakładzie pracy.

Kolejna różnica pomiędzy opisanymi powyżej typami testów polega na tym, że testowanie statyczne może być wykorzystywane do zwiększania spójności i wewnętrznej jakości produktów pracy, natomiast testowanie dynamiczne koncentruje się głównie na widocznych z zewnątrz zachowaniach systemu/oprogramowania.

Przykładami typowych defektów, które są łatwiejsze i tańsze do wykrycia oraz usunięcia przy zastosowaniu metody testowania statycznego (niż przy użyciu metody testowania dynamicznego) są między innymi:

- defekty w wymaganiach (takie jak: niespójności, niejednoznaczności, wewnętrzne sprzeczności, opuszczenia, nieścisłości, przeoczenia czy elementy nadmiarowe w wymaganiach);
- defekty w projekcie (np. nieefektywne algorytmy lub struktury baz danych, wysoki stopień sprzężenia (ang. *coupling*) czy mała spójność (ang. *cohesion*);
- defekty w kodzie (np. zmienne z niezdefiniowanymi wartościami, zmienne zadeklarowane — lecz nigdy nie używane, niedostępny (martwy) kod, powielony kod);
- odchylenia od standardów (np. brak zgodności ze standardami tworzenia kodu);
- niepoprawne specyfikacje interfejsów (np. użycie różnych jednostek miary w systemie wywołującym i systemie wywoływanym);
- słabe punkty zabezpieczeń (np. podatność na przepełnienie bufora);
- luki lub nieścisłości w zakresie śledzenia powiązań między podstawą testów a produktami pracy związanymi z testowaniem lub luki pokrycia (np. brak testów odpowiadających kryteriom akceptacji).

Ponadto tylko testowanie statyczne umożliwia wykrycie większości typów defektów związanych z utrzymywalnością (takich jak: nieprawidłowa modularyzacja, niski poziom ponownego wykorzystania modułów czy występowanie kodu, który trudno jest analizować i modyfikować bez wprowadzania nowych defektów).

3.2. Proces przeglądu

Przeglądy mogą mieć różny charakter: od nieformalnego po formalny. Cechą charakterystyczną przeglądów nieformalnych jest to, że nie przebiegają one zgodnie ze zdefiniowanym procesem, a uzyskanych dzięki nim informacji nie trzeba formalnie dokumentować. Z kolei przeglądy formalne są przeprowadzane zgodnie z udokumentowanymi procedurami i z udziałem zespołu o ustalonym wcześniej składzie, a ich rezultaty muszą być obowiązkowo dokumentowane. Stopień sformalizowania przeglądu zależy od takich czynników jak: model cyklu życia oprogramowania, dojrzałość procesu wytwarzania oprogramowania, złożoność produktu pracy będącego przedmiotem przeglądu, wymogi prawne czy konieczność prowadzenia ścieżki audytu.

To, na czym koncentruje się przegląd, zależy od uzgodnionych celów przeglądu, takich jak: wykrywanie defektów, poszerzanie wiedzy, edukowanie uczestników (np. testerów i nowych członków zespołu) bądź omawianie określonych zagadnień i podejmowanie decyzji w drodze konsensusu.

Bardziej szczegółowy opis procesu przeglądu produktów pracy (w tym ról i technik związanych z przeglądem) zawiera międzynarodowy standard ISO/IEC 20246.

3.2.1. Proces przeglądu produktów pracy

Proces przeglądu obejmuje zwykle następujące czynności:

Planowanie

- Określenie zakresu prac, w tym celu przeglądu i dokumentów (lub ich części) będących jego przedmiotem oraz ocenianych charakterystyk jakościowych.
- Oszacowanie nakładu pracy i ram czasowych.

- Wskazanie cech charakterystycznych przeglądu, takich jak: typ przeglądu, role, czynności i listy kontrolne.
- Wybór osób, które mają wziąć udział w przeglądzie oraz wyznaczenie im ról.
- Określenie kryteriów wejścia i wyjścia (w przypadku bardziej formalnych typów przeglądów np. inspekcji).
- Sprawdzenie, czy zostały spełnione kryteria wejścia (w przypadku bardziej formalnych typów przeglądów).

Rozpoczęcie przeglądu

- Rozesłanie produktu pracy (w formie papierowej lub elektronicznej) oraz innych materiałów (np.: formularzy dziennika problemów, list kontrolnych i innych powiązanych produktów pracy).
- Wyjaśnienie uczestnikom zakresu, celów, procesu, ról i produktów pracy.
- Udzielenie odpowiedzi na ewentualne pytania uczestników dotyczące przeglądu.

Przegląd indywidualny (tj. indywidualne przygotowanie)

- Dokonanie przeglądu całości lub części produktów pracy.
- Odnotowanie potencjalnych defektów oraz zaleceń i pytań.

Przekazanie informacji o problemach i analiza problemów

- Przekazanie informacji o zidentyfikowanych, potencjalnych defektach (np. na spotkaniu przeglądowym).
- Przeanalizowanie defektów (w tym wyznaczenie osób za nie odpowiedzialnych oraz określenie statusu defektów).
- Dokonanie oceny i udokumentowanie charakterystyk jakościowych.
- Dokonanie oceny wniosków z przeglądu pod kątem kryteriów wyjścia w celu podjęcia decyzji (odrzucenie produktu pracy, wymagane poważne zmiany, akceptacja - być może z niewielkimi zmianami).

Usunięcie defektów i raportowanie

- Utworzenie raportów o defektach dotyczących wykrytych usterek wymagających wprowadzenia zmian w produkcie pracy.
- Usunięcie defektów wykrytych w przeglądanym produkcie pracy (czynność tę zwykle wykonuje autor).
- Poinformowanie odpowiedniej osoby lub odpowiedniego zespołu o defektach wykrytych w produkcie pracy związanym z produktem będącym przedmiotem przeglądu.
- Zarejestrowanie zaktualizowanego statusu defektów (w przypadku przeglądów formalnych), włączając w to potencjalną zgodę autora komentarza.
- Zebranie miar (w przypadku bardziej formalnych typów przeglądów).
- Sprawdzenie, czy zostały spełnione kryteria wyjścia (w przypadku bardziej formalnych typów przeglądów).

- Zaakceptowanie produktu pracy po spełnieniu kryteriów wyjścia.

Rezultaty przeglądu produktu pracy mogą być różne w zależności od typu przeglądu oraz stopnia jego sformalizowania (patrz p. 3.2.3.).

3.2.2. Role i obowiązki w przeglądzie formalnym

W typowym przeglądzie formalnym wyróżnia się następujące role:

Autor

- Tworzy produkt pracy będący przedmiotem przeglądu.
- Usuwa defekty w produkcie pracy będącym przedmiotem przeglądu (jeśli jest to konieczne).

Kierownictwo

- Odpowiada za zaplanowanie przeglądu.
- Podejmuje decyzję o przeprowadzeniu przeglądu.
- Wyznacza pracowników oraz określa budżet i ramy czasowe.
- Monitoruje na bieżąco opłacalność przeglądu.
- Wykonuje decyzje kontrolne w przypadku uzyskania niezadowalających wyników.

Facylitator (zwany często moderatorem)

- Dba o sprawny przebieg spotkań związanych z przeglądem (o ile przeglądy się odbywają)
- Występuje w roli mediatora, jeśli konieczne jest pogodzenie różnych punktów widzenia.
- Jest często osobą, od której zależy powodzenie przeglądu.

Lider przeglądu

- Ponosi ogólną odpowiedzialność za przegląd.
- Decyduje o tym, kto ma wziąć udział w przeglądzie oraz określa miejsce i termin przeglądu.

Przeglądający

- Mogą być ekspertami merytorycznymi, osobami pracującymi przy projekcie, interesariuszami zainteresowanymi produktem pracy i/lub osobami mającymi określone doświadczenie techniczne lub biznesowe.
- Identyfikują potencjalne defekty w przeglądanym produkcie pracy
- Mogą reprezentować różne punkty widzenia (np. punkt widzenia testera, dewelopera, użytkownika, operatora, analityka biznesowego, specjalisty ds. użyteczności itd.).

Protokolant

- Gromadzi informacje o potencjalnych defektach wykrytych w ramach przeglądu indywidualnego.
- Rejestruje nowe potencjalne defekty, otwarte punkty i decyzje podczas spotkania związanego z przeglądem (gdy ma ono miejsce).

W przypadku niektórych typów przeglądów jedna osoba może pełnić kilka ról. W zależności od typu przeglądu mogą również różnić się czynności przypisane do poszczególnych ról. Ponadto w związku z pojawieniem się narzędzi wspomagających proces przeglądu (a zwłaszcza rejestrowanie defektów, otwartych punktów i decyzji), często nie ma potrzeby wyznaczania protokolanta.

Oprócz ról opisanych powyżej mogą również występować bardziej szczegółowe role opisane w standardzie ISO/IEC 20246.

3.2.3. Typy przeglądów

Przeglądy mogą być przeprowadzane w różnych celach, ale jednym z głównych celów każdego przeglądu jest ujawnienie defektów. W wykryciu defektów mogą pomóc przeglądy wszystkich typów. Odpowiedni typ przeglądu należy wybrać na podstawie potrzeb projektu, dostępnych zasobów, typu produktu i związanych z nim czynników ryzyka, dziedziny biznesowej, kultury korporacyjnej i innych kryteriów.

Produkt pracy może być przedmiotem więcej niż jednego typu przeglądu, a w przypadku przeprowadzania kilku przeglądów różnych typów ich kolejność może być różna — np. przed przeglądem technicznym może zostać przeprowadzony przegląd nieformalny, by upewnić się, że produkt pracy jest gotowy do przeglądu technicznego.

Wszystkie opisane poniżej typy przeglądów mogą być realizowane jako przeglądy koleżeńskie, czyli przeprowadzane przez współpracowników pracujących na zbliżonym szczeblu organizacyjnym.

Typy defektów znalezionych w przeglądach są różne, zależą w szczególności od przeglądanego produktu pracy. Przykłady defektów, które można znaleźć w przeglądach różnych produktów pracy, zostały opisane w p. 3.1.3., zaś informacje na temat inspekcji formalnych można znaleźć w [Gilb 1993].

Przeglądy mogą być klasyfikowane według różnych atrybutów. Poniżej wymieniono cztery najczęściej występujące typy przeglądów wraz z odpowiadającymi im atrybutami.

Przegląd nieformalny (np. sprawdzenie koleżeńskie, przegląd w parach)

- Cel główny: wykrycie potencjalnych defektów.
- Możliwy cel dodatkowy: wygenerowanie nowych pomysłów lub rozwiązań, szybkie rozwiązywanie problemów mniejszej wagi.
- Przegląd nie odbywa się zgodnie z formalnym (udokumentowanym) procesem.
- Przegląd nie musi obejmować spotkania.
- Przegląd może zostać przeprowadzony przez współpracownika autora (sprawdzenie koleżeńskie) lub przez grupę osób.
- Rezultaty przeglądu mogą zostać udokumentowane.
- Przydatność przeglądu zależy od przeglądających.

- Używanie list kontrolnych jest opcjonalne.
- Ten typ przeglądu jest powszechnie używany w przypadku zwinnego wytwarzania oprogramowania.

Przejrzenie

- Cele główne: wykrycie potencjalnych defektów, podniesienie jakości oprogramowania, rozważenie alternatywnych implementacji, dokonanie oceny zgodności z normami/standardami i specyfikacjami.
- Możliwe cele dodatkowe: wymiana informacji na temat technik lub odmian stylów, przeszkolenie uczestników, osiągnięcie konsensusu.
- Indywidualne przygotowanie przed spotkaniem związanym z przeglądem jest opcjonalne.
- Spotkanie związane z przeglądem prowadzi zwykle autor produktu pracy.
- Wymagany jest udział protokolanta.
- Używanie list kontrolnych jest opcjonalne.
- Przegląd może mieć formę scenariuszy, przebiegów próbnych („na sucho") lub symulacji.
- Dla tego typu przeglądu tworzone są: dziennik (log) potencjalnych defektów i raporty z przeglądu.
- Charakter przeglądu może być w praktyce różny — od raczej nieformalnego po bardzo formalny.

Przegląd techniczny

- Cele główne: uzyskanie konsensusu, wykrycie potencjalnych defektów.
- Możliwe cele dodatkowe: dokonanie oceny jakości produktu pracy i zwiększenie zaufania do niego, wygenerowanie nowych pomysłów, zmotywowanie autorów do udoskonalania przyszłych produktów pracy i stworzenie im warunków do tego, rozważenie alternatywnych rozwiązań.
- Przeglądający powinni mieć takie same kompetencje techniczne jak autor oraz powinni być ekspertami technicznymi w tej samej dyscyplinie lub w innych dyscyplinach.
- Indywidualne przygotowanie przed spotkaniem związanym z przeglądem jest obowiązkowe.
- Spotkanie związane z przeglądem jest opcjonalne, powinien je w miarę możliwości prowadzić przeszkolony facylitator (zwykle nie jest nim autor).
- Wymagany jest udział protokolanta (w miarę możliwości nie powinien nim być autor).
- Korzystanie z list kontrolnych jest opcjonalne.
- Dla tego typu przeglądu tworzone są: dziennik (log) potencjalnych defektów i raporty z przeglądu.

Inspekcja

- Cele główne: wykrycie potencjalnych defektów, dokonanie oceny jakości produktu pracy i zwiększenie zaufania do niego, zapobieganie wystąpieniu w przyszłości podobnych defektów poprzez przekazanie wniosków autorowi i przeprowadzenie analizy przyczyny podstawowej.
- Możliwe cele dodatkowe: zmotywowanie autorów do udoskonalania przyszłych produktów pracy oraz procesu wytwarzania oprogramowania i stworzenie im warunków do tego, osiągnięcie konsensusu.
- Przegląd odbywa się zgodnie ze zdefiniowanym procesem opartym na regułach i listach kontrolnych, a rezultaty są formalnie dokumentowane.
- Występują jednoznacznie określone role (takie jak role określone w p. 3.2.2.), a ponadto może być wymagane wyznaczenie konkretnej osoby czytającej (która głośno czyta produkt pracy, często parafrazując tj. opisując produkt własnymi słowami, podczas spotkania przeglądowego).
- Indywidualne przygotowanie przed spotkaniem związanym z przeglądem jest obowiązkowe.
- Przeglądającymi są osoby zajmujące stanowiska równorzędne do autora lub eksperci w innych dyscyplinach istotnych z punktu widzenia produktu pracy.
- Obowiązują określone kryteria wejścia i wyjścia.
- Obecność protokolanta jest obowiązkowa.
- Przegląd prowadzi przeszkolony facylitator (nie jest nim autor).
- Autor nie może być liderem przeglądu, osobą czytającą ani protokolantem.
- Mogą być tworzone dzienniki potencjalnych defektów i raporty z przeglądu.
- Zbierane są miary wykorzystywane następnie do udoskonalania całego procesu wytwarzania oprogramowania (w tym procesu inspekcji).

3.2.4. Stosowanie technik przeglądu

Istnieje kilka różnych technik, z których można skorzystać podczas przeglądu indywidualnego (tj. indywidualnego przygotowania) przeprowadzanego w celu wykrycia defektów. Techniki te można stosować w ramach wszystkich opisanych powyżej typów przeglądów, jednak ich skuteczność może różnić się w zależności od wybranego typu przeglądu. Poniżej przedstawiono przykładowe techniki przeglądu indywidualnego w kontekście różnych typów przeglądów.

Przegląd *ad hoc*

Podczas przeglądu *ad hoc* przeglądający otrzymują niewiele wskazówek dotyczących sposobu wykonywania zadania (lub nie otrzymują ich wcale). Uczestnicy często czytają produkt pracy sekwencyjnie, na bieżąco identyfikując i dokumentując stwierdzone problemy. Przegląd *ad hoc* to powszechnie stosowana technika, której stosowanie nie wymaga intensywnych przygotowań. Powodzenie przeglądu zależy w dużej mierze od umiejętności przeglądających. Ponadto technika ta wiąże się z ryzykiem wielokrotnego zgłaszania tych samych problemów przez różnych przeglądających.

Przegląd oparty na liście kontrolnej

Przegląd oparty na liście kontrolnej to usystematyzowana technika, zgodnie z którą przeglądający wykrywają problemy na podstawie list kontrolnych rozsyłanych w momencie rozpoczęcia przeglądu (np. przez facylitatora). Lista kontrolna przeglądu zawiera zestaw pytań odpowiadających potencjalnym defektom wskazanym na podstawie dotychczasowych doświadczeń. Listy kontrolne powinny być dostosowane do specyfiki przeglądanego produktu pracy i regularnie aktualizowane w celu uwzględnienia typów problemów, które zostały pominięte w poprzednich przeglądach. Najważniejszą zaletą techniki opartej na listach kontrolnych jest systematyczne pokrycie najczęściej występujących typów defektów. Należy jednak pamiętać, że przegląd indywidualny nie powinien ograniczać się do wykonania czynności z listy kontrolnej. Ważne jest również szukanie defektów, które nie zostały w niej uwzględnione.

Scenariusze i przebiegi próbne

Podczas przeglądu opartego na scenariuszach przeglądający otrzymują ustrukturyzowane wytyczne dotyczące sposobu zapoznawania się z produktem pracy. Przegląd oparty na scenariuszach umożliwia przeglądającym wykonywanie przebiegów próbnych takich produktów zgodnie z ich przewidywanym zastosowaniem (jeśli produkty pracy są udokumentowane w odpowiednim formacie np. jako przypadki użycia). Dzięki scenariuszom przeglądający mogą skuteczniej identyfikować określone typy defektów niż przy użyciu prostych list kontrolnych. Podobnie jak w przypadku przeglądów opartych na listach kontrolnych, przeglądający nie powinni ograniczać się do udokumentowanych scenariuszy, ponieważ grozi to pominięciem innych typów defektów (takich jak brakujące funkcjonalności).

Czytanie oparte na perspektywie

W przypadku czytania opartego na perspektywie, podobnie jak w przypadku przeglądu opartego na rolach (omówionego poniżej), przeglądający przyjmują podczas przeglądu indywidualnego punkty widzenia różnych interesariuszy (zwykle są to punkty widzenia użytkownika, pracownika działu marketingu, projektanta, testera lub operatora). Uwzględnienie punktów widzenia różnych interesariuszy umożliwia bardziej wnikliwe przeprowadzenie przeglądu indywidualnego, a także pozwala uniknąć wielokrotnego zgłaszania tych samych problemów przez różnych przeglądających.

Ponadto czytanie oparte na perspektywie wymaga również od przeglądających, aby spróbowali wykorzystać produkt pracy podlegający przeglądowi do wygenerowania produktu, który się da z niego wyprowadzić. Przykładem takiego wykorzystania produktu pracy jest podjęcie przez testera próby wygenerowania wersji roboczych testów akceptacyjnych, jeśli przeprowadzi przegląd specyfikacji wymagań oparty na perspektywie, aby sprawdzić, czy zawiera wszystkie niezbędne informacje. Ponadto w czytaniu opartym na perspektywie oczekuje się użycia list kontrolnych.

Badania empiryczne wykazały, że czytanie oparte na perspektywie jest zazwyczaj najskuteczniejszą techniką przeglądu wymagań i produktów pracy o charakterze technicznym. Kluczowym czynnikiem sukcesu jest uwzględnienie i wyważenie punktów widzenia poszczególnych interesariuszy w sposób adekwatny do ryzyka.

Więcej szczegółowych informacji na temat czytania opartego na perspektywie można znaleźć w [Shul 2000], a informacji na temat skuteczności różnych typów przeglądów w [Sauer 2000].

Przegląd oparty na rolach

Przegląd oparty na rolach to technika, w której przeglądający oceniają produkt pracy z perspektywy poszczególnych ról przypisanych interesariuszom. Typowe role obejmują konkretne typy użytkowników końcowych (doświadczonych, niedoświadczonych, starszych, dzieci itp.) oraz określone funkcje w organizacji (użytkownik, administrator, administrator systemu, tester wydajności itp.).

W przeglądzie tym obowiązują te same zasady, co w przypadku czytania opartego na perspektywie, ponieważ role są podobne.

3.2.5. Czynniki powodzenia związane z przeglądami

Kluczem do pomyślnego przeprowadzenia przeglądu jest staranny dobór typu przeglądu i stosowanych technik. Ponadto należy uwzględnić wiele innych czynników, które mogą mieć wpływ na uzyskany wynik przeglądu.

Czynniki sukcesu o charakterze organizacyjnym to:

- każdy przegląd ma jednoznaczne cele, które zdefiniowane zostały podczas planowania i które wykorzystuje się jako mierzalne kryteria wyjścia;
- stosowane typy przeglądów sprzyjają osiągnięciu założonych celów, a także są adekwatne do typu i poziomu produktów pracy związanych z oprogramowaniem oraz do poziomu wiedzy i doświadczenia uczestników;
- stosowane są różne techniki przeglądu (takie jak przegląd oparty na listach kontrolnych lub oparty na rolach), które pozwalają skutecznie identyfikować defekty występujące w danym produkcie pracy;
- stosowane listy kontrolne są aktualne i uwzględniają główne czynniki ryzyka;
- obszerniejsze dokumenty są pisane i przeglądane partiami, dzięki czemu autorzy szybko i często otrzymują informacje zwrotne na temat defektów (co jest elementem kontroli jakości);
- uczestnicy mają wystarczająco dużo czasu na indywidualne przygotowanie;
- przeglądy są planowane z należytym wyprzedzeniem;
- kierownictwo wspiera przebieg przeglądów (np. poprzez wyznaczenie wystarczającej ilości czasu na wykonanie czynności związanych z przeglądem w harmonogramie projektu).
- przeglądy są spójne z polityką jakości i/lub polityką testów w organizacji.

Czynniki sukcesu o charakterze kadrowym to::

- w przegląd są zaangażowane osoby, których udział sprzyja osiągnięciu jego celów — np. osoby o różnych umiejętnościach lub różnych punktach widzenia, które będą potencjalnie korzystać z przeglądanego dokumentu w ramach wykonywanej pracy;
- testerzy są uznawani za ważnych uczestników przeglądu, a zdobywana przez nich wiedza na temat produktu pracy umożliwia wcześniejsze przygotowanie bardziej efektywnych testów;
- uczestnicy przeznaczają wystarczającą ilość czasu na wzięcie udziału w przeglądzie i wykazują się należytą dbałością o szczegóły;
- przeglądane są małe fragmenty dokumentacji, dzięki czemu przeglądający nie tracą koncentracji podczas pracy indywidualnej i/lub spotkania przeglądowego (o ile się odbywa);
- informacje o wykrytych defektach są przyjmowane do wiadomości, zaś defekty są potwierdzane i rozpatrywane w sposób obiektywny;
- spotkanie jest prowadzone we właściwy sposób, tak, aby uczestnicy mieli poczucie, że pożytecznie spędzili czas;

- przegląd odbywa się w atmosferze wzajemnego zaufania, a jego wyniki nie są wykorzystywane do oceny uczestników przeglądu;
- uczestnicy unikają gestów i zachowań, które mogłyby wskazywać na znudzenie, irytację bądź wrogie nastawienie wobec innych uczestników;
- uczestnikom zapewnia się należyte szkolenie, zwłaszcza w przypadku bardziej formalnych typów przeglądów (takich jak inspekcje);
- tworzy się atmosferę sprzyjającą poszerzaniu wiedzy i doskonaleniu procesów.

Więcej informacji o udanych przeglądach można znaleźć w [Gilb 1993], [Wiegers 2002] i [van Veenendaal 2004].

4. Techniki testowania — 330 min.

Słowa kluczowe

analiza wartości brzegowych, białoskrzynkowae technika testowania, czarnoskrzynkowa technika testowania, podział na klasy równoważności, pokrycie, pokrycie decyzji, pokrycie instrukcji kodu, technika testowania, technika testowania oparte na doświadczeniu, testowanie eksploracyjne, testowanie oparte na przypadkach użycia, testowanie przejść pomiędzy stanami, testowanie w oparciu o listę kontrolną, testowanie w oparciu o tablicę decyzyjną, zgadywanie błędów

Cele nauczania — techniki testowania

4.1. Kategorie technik testowania

FL-4.1.1. (K2) Kandydat potrafi wyjaśnić cechy charakterystyczne i elementy wspólne czarnoskrzynkowych technik testowania, białoskrzynkowych technik testowania oraz technik testowania opartych na doświadczeniu, a także różnice między nimi.

4.2. Czarnoskrzynkowe techniki testowania

FL-4.2.1. (K3) Kandydat potrafi zaprojektować przypadki testowe na podstawie podanych wymagań metodą podziału na klasy równoważności.

FL-4.2.2. (K3) Kandydat potrafi zaprojektować przypadki testowe na podstawie podanych wymagań metodą analizy wartości brzegowych.

FL-4.2.3. (K3) Kandydat potrafi zaprojektować przypadki testowe na podstawie podanych wymagań metodą testowania w oparciu o tablicę decyzyjną.

FL-4.2.4. (K3) Kandydat potrafi zaprojektować przypadki testowe na podstawie podanych wymagań metodą testowania przejść pomiędzy stanami.

FL-4.2.5. (K2) Kandydat potrafi wyjaśnić, w jaki sposób można wyprowadzać przypadki testowe z przypadku użycia.

4.3. Białoskrzynkowe techniki testowania

FL-4.3.1. (K2) Kandydat potrafi wyjaśnić pojęcie pokrycia instrukcji kodu.

FL-4.3.2. (K2) Kandydat potrafi wyjaśnić pojęcie pokrycia decyzji.

FL-4.3.3. (K2) Kandydat potrafi wyjaśnić korzyści wynikające z pokrycia instrukcji kodu i pokrycia decyzji.

4.4. Techniki testowania oparte na doświadczeniu

FL-4.4.1. (K2) Kandydat potrafi wyjaśnić pojęcie zgadywania błędów.

FL-4.4.2. (K2) Kandydat potrafi wyjaśnić pojęcie testowania eksploracyjnego.

FL-4.4.3. (K2) Kandydat potrafi wyjaśnić pojęcie testowania w oparciu o listę kontrolną.

4.1. Kategorie technik testowania

Każda technika testowania, włączając te omówione w bieżącym rozdziale, ma z założenia pomagać w identyfikowaniu warunków testowych, przypadków testowych i danych testowych.

Przy wyborze technik testowania, które będą stosowane, należy wziąć pod uwagę wiele czynników takich jak:

- złożoność modułu lub systemu;
- obowiązujące przepisy i normy;
- wymagania klienta lub wymagania wynikające z umów;
- poziomy ryzyka i jego rodzaj;
- dostępną dokumentację;
- wiedzę i umiejętności testerów;
- dostępne narzędzia;
- czas i budżet;
- model cyklu życia oprogramowania;
- typy defektów spodziewane w modułach i systemach.

Niektóre techniki testowania lepiej sprawdzają się w określonych sytuacjach lub na określonych poziomach testów, inne zaś można z powodzeniem stosować na każdym poziomie testów. Aby uzyskać najlepsze rezultaty, testerzy zazwyczaj tworzą przypadki testowe łącząc ze sobą różne techniki.

Użycie technik testowania w analizie, projektowaniu i implementacji testów może mieć zróżnicowany charakter: od bardzo nieformalnego (wymagającego minimalnej lub niewymagającego żadnej dokumentacji) po bardzo formalny. Właściwy stopień sformalizowania zależy od kontekstu testowania, w tym od: dojrzałości procesów testowych i procesów wytwarzania oprogramowania, ograniczeń czasowych, norm bezpieczeństwa i wymogów prawnych, wiedzy i umiejętności zaangażowanych osób oraz przyjętego modelu cyklu życia oprogramowania.

4.1.1. Kategorie technik testowania i ich cechy charakterystyczne

W niniejszym sylabusie techniki testowania podzielono na: czarnoskrzynkowe, białoskrzynkowe i oparte na doświadczeniu.

Techniki czarnoskrzynkowe (zwane również technikami behawioralnymi lub opartymi na specyfikacji) bazują na analizie podstawy testów np. formalnych dokumentach zawierających wymagania, specyfikacje, przypadki użycia, historyjki użytkownika lub opis procesów biznesowych. Techniki z tej grupy można stosować zarówno w testowaniu funkcjonalnym, jak i w testowaniu niefunkcjonalnym. Techniki czarnoskrzynkowe koncentrują się na danych wejściowych i wyjściowych przedmiotu testów, bez odwoływania się do jego struktury wewnętrznej.

Podstawą białoskrzynkowych technik testowania (zwanych także technikami strukturalnymi lub opartymi na strukturze) jest analiza architektury, szczegółowego projektu, struktury wewnętrznej lub kodu przedmiotu testów. W przeciwieństwie do technik czarnoskrzynkowych, białoskrzynkowe techniki testowania koncentrują się na strukturze i przetwarzaniu wewnątrz przedmiotu testów.

Techniki testowania oparte na doświadczeniu pozwalają wykorzystać doświadczenie deweloperów, testerów i użytkowników do projektowania, implementowania i wykonywania testów. Techniki te są często stosowane razem z technikami czarnoskrzynkowymi i białoskrzynkowymi.

Najważniejsze cechy charakterystyczne czarnoskrzynkowych technik testowania:

- warunki testowe, przypadki testowe i dane testowe wyprowadza się z podstawy testów, którą mogą stanowić wymagania na oprogramowanie, specyfikacje, przypadki użycia i historyjki użytkownika;
- przypadki testowe mogą być wykorzystywane do wykrywania rozbieżności między wymaganiami a ich implementacją bądź odstępstw od wymagań;
- pokrycie mierzy się biorąc pod uwagę przetestowane elementy podstawy testów i technikę zastosowaną do podstawy testów.

Najważniejsze cechy charakterystyczne białoskrzynkowych technik testowania:

- warunki testowe, przypadki testowe i dane testowe wyprowadza się z podstawy testów, która może obejmować: kod, architekturę oprogramowania, szczegółowy projekt bądź dowolne inne źródło informacji o strukturze oprogramowania;
- pokrycie mierzy się biorąc pod uwagę przetestowane elementów wybranej struktury (np. kod lub interfejsy) oraz zastosowaną technikę do podstawy testów.

Najważniejsze cechy charakterystyczne technik testowania opartych na doświadczeniu:

- warunki testowe, przypadki testowe i dane testowe wyprowadza się z podstawy testów, która może obejmować wiedzę i doświadczenie testerów, deweloperów, użytkowników oraz innych interesariuszy;
- wiedza i doświadczenie mogą dotyczyć między innymi przewidywanego sposobu korzystania z oprogramowania, środowiska pracy oprogramowania oraz prawdopodobnych defektów i ich rozkładu.

Techniki testowania i odpowiadające im miary pokrycia opisuje międzynarodowy standard ISO/IEC/IEEE 29119-4. Więcej informacji o technikach testowania zawierają publikacje [Craig 2002] oraz [Copeland 2004], a także [Roman 2015].

4.2. Czarnoskrzynkowe techniki testowania

4.2.1. Podział na klasy równoważności

Technika podziału na klasy równoważności polega na dzieleniu danych na grupy (zwane klasami równoważności lub klasami abstrakcji) w taki sposób, aby każda grupa zawierała elementy, które z założenia mają być przetwarzane w ten sam sposób (zobacz [Kaner 2013] i [Jorgensen 2014], a także [Roman 2015]).

Klasy równoważności można wyznaczać zarówno dla wartości poprawnych, jak i dla wartości niepoprawnych:

- wartości poprawne to wartości, które powinny zostać zaakceptowane przez moduł lub system, a zawierająca je klasa równoważności nosi nazwę „poprawnej klasy równoważności”;
- wartości niepoprawne to wartości, które moduł lub system powinien odrzucić, a zawierająca je klasa równoważności to „niepoprawna klasa równoważności”;
- klasy można identyfikować w odniesieniu do wszelkich elementów danych, które są związane z przedmiotem testów, takich jak: dane wejściowe, dane wyjściowe, wartości wewnętrzne bądź wartości zależne od czasu (np. występujące przed zdarzeniem lub po zdarzeniu), a także w odniesieniu do parametrów interfejsu (np. integrowanych modułów testowanych w ramach testowania integracyjnego);
- każdą klasę można w razie potrzeby podzielić na podklasy;
- każda wartość musi należeć do jednej i tylko jednej klasy równoważności;
- jeśli w przypadkach testowych są stosowane niepoprawne klasy równoważności, należy je testować indywidualnie (tzn. nie należy ich łączyć z innymi niepoprawnymi klasami równoważności[3]), aby uniknąć zamaskowania awarii. Maskowanie awarii może mieć miejsce, gdy w tej samej chwili występuje kilka awarii, ale tylko jedna jest widoczna, przez co pozostałe awarie pozostają niewykryte.;

Warunkiem uzyskania stuprocentowego pokrycia przy korzystaniu z tej techniki jest pokrycie przez przypadki testowe wszystkich zidentyfikowanych klas równoważności (włączając niepoprawne klasy równoważności), co wymaga wybrania do testów co najmniej jednej wartości z każdej klasy. Pokrycie mierzy się jako iloraz liczby klas równoważności przetestowanych przy użyciu co najmniej jednej wartości przez łączną liczbę zdefiniowanych klas równoważności i zazwyczaj wyrażane jest w procentach. Podział na klasy równoważności można stosować na wszystkich poziomach testów.

4.2.2. Analiza wartości brzegowych

Technika analizy wartości brzegowych jest rozszerzeniem techniki podziału na klasy równoważności, ale może być stosowana tylko w przypadku uporządkowanych klas zawierających dane liczbowe lub sekwencyjne. Wartościami brzegowymi klasy równoważności są jej wartość minimalna i maksymalna (lub pierwsza i ostatnia wartość) [Beizer 1990].

Rozważmy następujący przykład. Załóżmy, że pole umożliwia akceptację jednoznakowej wartości całkowitej wprowadzanej z klawiatury numerycznej (tym samym zakładamy, że niemożliwe jest wprowadzenie wartości nienumerycznych). Akceptowalny zakres wartości całkowitych mieści się przedziale od 1 do 5 włącznie. W tym przypadku możemy więc wyróżnić trzy klasy równoważności: niepoprawna (wartości za niskie), poprawna, niepoprawna (wartości za wysokie). Dla poprawnej klasy równoważności wartości brzegowe to 1 i 5. Dla niepoprawnej klasy równoważności (za wysokie) wartość brzegowa to 6. Dla niepoprawnej klasy równoważności (za niskie) jest tylko jedna wartość brzegowa 0, ponieważ jest to klasa z jednym tylko przedstawicielem.

W powyższym przykładzie identyfikujemy dwie wartości brzegowe do przetestowania dla każdej granicy pomiędzy sąsiadującymi klasami. Granica pomiędzy klasą niepoprawną (za niskie) a poprawną daje dane testowe 0 i 1. Granica pomiędzy klasą poprawną i niepoprawną (za wysokie) daje dane testowe 5

[3] Niniejszy fragment sylabusa dotyczy sytuacji, w której przypadek testowy zawiera rozważane jednocześnie wartości z kilku klas równoważności pochodzących z *różnych* podziałów jednej lub kilku dziedzin (przyp. tłum.).

i 6 Pewna odmiana tej techniki identyfikuje trzy wartości do przetestowania dla zadanej wartości brzegowej: wartość leżącą tuż przed wartością brzegową, wartość brzegową i wartość leżącą tuż za wartością brzegową. Jeśli rozpatrujemy poprawną klasę równoważności z poprzedniego przykładu, wartości testowe dla dolnej wartości brzegowej wynoszą 0, 1 i 2, a wartości testowe dla górnej wartości brzegowej — 4, 5 i 6 [Jorgensen 2014].

Niepoprawne zachowanie jest bardziej prawdopodobne dla wartości brzegowych klas równoważności niż dla wartości z wnętrza klasy. Należy pamiętać, że zarówno wyspecyfikowane, jak i zaimplementowane wartości brzegowe, mogą zostać przesunięte powyżej lub poniżej zamierzonego położenia lub całkowicie pominięte, a ponadto mogą wystąpić niechciane nadmiarowe wartości brzegowe. Analiza wartości brzegowych i ich testowanie pozwala odkryć niemal wszystkie takie defekty, ponieważ pozwala na wykazanie w oprogramowaniu zachowań z klasy równoważności innej niż klasa, do której powinna należeć wartość brzegowa.

Analizę wartości brzegowych można stosować na wszystkich poziomach testów. Technika ta służy zwykle do testowania wymagań, które odwołują się do przedziału liczb (włączając w to daty i godziny). Pokrycie wartości brzegowych danej klasy mierzy się jako iloraz liczby przetestowanych wartości brzegowych przez łączną liczbę zidentyfikowanych wartości brzegowych, zazwyczaj wyrażony w procentach[4].

4.2.3. Testowanie w oparciu o tablicę decyzyjną

Tablice decyzyjne są dobrym sposobem na modelowanie złożonych reguł biznesowych, które muszą zostać zaimplementowane w systemie. Opracowując tablice decyzyjne, tester identyfikuje warunki (zazwyczaj dane wejściowe) i wynikające z nich akcje (zazwyczaj dane wyjściowe) systemu. Tworzą one wiersze tablicy, przy czym warunki znajdują się zwykle u góry, a akcje — na dole. Każda kolumna odpowiada regule decyzyjnej, która określa unikatową kombinację warunków powodującą wykonanie akcji związanych z tą regułą. Wartości warunków i akcji przedstawia się zwykle jako wartości logiczne (prawda/fałsz) lub dyskretne (np. czerwony, zielony, niebieski), ale mogą one mieć również postać liczb lub przedziałów liczb. W tej samej tablicy mogą występować różne typy warunków i akcji.

Poniżej przedstawiono typową notację stosowaną w tablicach decyzyjnych.

Warunki:

- „T” oznacza, że warunek został spełniony (spotykane są również oznaczenia „P” i „1”).
- „N” oznacza, że warunek nie został spełniony (spotykane są również oznaczenia „F” i „0”).
- „—” oznacza, że wartość warunku nie ma znaczenia (spotykane jest również oznaczenie „nd”).

Akcje:

- „X” oznacza, że akcja powinna zostać wykonana (spotykane są również oznaczenia „T”, „P” i „1”).
- Puste pole oznacza, że akcja nie powinna zostać wykonana (spotykane są również oznaczenia „N”, „F” i „0”).

Pełna tablica decyzyjna zawiera tyle kolumn (przypadków testowych), ile jest niezbędne do pokrycia wszystkich kombinacji warunków. Usuwając kolumny, które nie wpływają na wynik testów, liczbę

[4] W przypadku wspomnianej wcześniej odmiany „trójpunktowej" metryka pokrycia uwzględnia nie tylko wartości brzegowe, ale również wszystkie wartości do przetestowania, mimo, że niektóre z nich formalnie nie są wartościami brzegowymi — np. wartości 2 i 4 z cytowanego wyżej przykładu (przyp. tłum.).

przypadków testowych można znacznie zmniejszyć, na przykład poprzez usunięcie niemożliwych do spełnienia kombinacji warunków. Więcej informacji na temat minimalizowania tablic decyzyjnych zawiera sylabus [ISTQB-CTAL-TA].

Minimalnym wymogiem dotyczącym pokrycia w przypadku testowania w oparciu o tablicę decyzyjną jest zazwyczaj utworzenie co najmniej jednego przypadku testowego dla każdej reguły decyzyjnej w tablicy. Zwykle wiąże się to z koniecznością pokrycia wszystkich kombinacji warunków. Pokrycie jest mierzone jako iloraz liczby reguł decyzyjnych przetestowanych przez przynajmniej jeden przypadek testowy przez łączną liczbę reguł decyzyjnych, zazwyczaj wyrażony w procentach.

Zaletą testowania w oparciu o tablicę decyzyjną jest możliwość zidentyfikowania wszystkich ważnych, istotnych kombinacji warunków, które w innym przypadku mogłyby zostać przeoczone. Ponadto metoda ta pomaga znaleźć ewentualne luki w wymaganiach. Można ją stosować we wszystkich sytuacjach, w których zachowanie oprogramowania zależy od kombinacji warunków oraz na dowolnym poziomie testów.

4.2.4. Testowanie przejść pomiędzy stanami

Moduł lub system mogą różnie reagować na dane wejściowe w zależności od warunków bieżących lub historycznych (np. zdarzeń, które wystąpiły od momentu uruchomienia systemu). Historię dotychczasowych zdarzeń można przedstawiać przy pomocy pojęcia stanu. Do zobrazowania możliwych stanów oprogramowania oraz przejść między nimi służy diagram przejść między stanami. Przejście inicjowane jest przez zdarzenie (np. wprowadzenie przez użytkownika wartości w pole). Wynikiem zdarzenia jest przejście ze stanu do stanu. To samo zdarzenie może skutkować dwoma lub więcej przejściami z tego samego stanu. Rezultatem zmiany stanu może być akcja oprogramowania (np. wyświetlenie wyniku lub komunikatu błędu).

Tablica przejść między stanami zawiera wszystkie poprawne przejścia między stanami (a potencjalnie również przejścia niepoprawne) oraz zdarzenia i akcje związane z poprawnymi przejściami. Diagramy przejść[5] między stanami przedstawiają zwykle tylko poprawne przejścia, nie zawierają natomiast przejść niepoprawnych.

Testy przejść między stanami można zaprojektować w sposób zapewniający pokrycie typowej sekwencji stanów, przetestowanie wszystkich stanów bądź przetestowanie wszystkich przejść, konkretnych sekwencji przejść lub przejść niepoprawnych.

Technikę testowania przejść między stanami stosuje się w przypadku aplikacji wyposażonych w menu. Jest ona również rozpowszechniona w branży oprogramowania wbudowanego. Ponadto technika ta nadaje się do modelowania scenariuszy biznesowych zawierających określone stany oraz do testowania sposobu poruszania się (nawigacji) po ekranach. Pojęcie stanu ma charakter abstrakcyjny i może odpowiadać zarówno kilku wierszom kodu, jak i całemu procesowi biznesowemu.

Pokrycie zwykle mierzy się jako iloraz liczby przetestowanych stanów lub przejść między stanami przez łączną liczbę zidentyfikowanych stanów lub przejść w przedmiocie testów, zazwyczaj wyrażony w procentach. Więcej informacji na temat kryteriów pokrycia dotyczących testowania przejść między stanami można znaleźć w [ISTQB-CTAL-TA], a także w [Roman 2015].

4.2.5. Testowanie oparte na przypadkach użycia

Testy można wyprowadzać z przypadków użycia. Opisują one interakcje z elementami oprogramowania, weryfikując wymagania dla funkcjonalności. Z przypadkami użycia związane są

[5] Diagram przejść jest graficzną prezentacją tablicy przejść (przyp. tłum.).

pojęcia „aktorów” (tj. użytkowników, urządzeń zewnętrznych bądź innych modułów lub systemów) oraz „podmiotów” (moduł lub system, do którego przypadek użycia jest zastosowany).

Każdy przypadek użycia określa konkretne działanie (zachowanie), które podmiot może wykonywać we współpracy z jednym lub kilkoma aktorami (UML 2.5.1 2017). Przypadki użycia można opisywać w kategoriach interakcji i działań, warunków wstępnych bądź warunków wyjściowych, a jeśli wymagają tego okoliczności — również w języku naturalnym. Interakcje między aktorami a podmiotem mogą powodować zmianę stanu podmiotu. Do odwzorowywania takich interakcji można użyć graficznych modeli przepływów pracy (ang. *workflow*), diagramów aktywności lub modeli procesów biznesowych.

Przypadek użycia może obejmować potencjalne odmiany podstawowego zachowania, zachowania wyjątkowe i mechanizmy obsługi błędów (odpowiedź systemu i ewentualnie odtworzenie go po wystąpieniu awarii wynikającej z pomyłki, defektu w aplikacji lub błędu komunikacji np. skutkującego komunikatem błędu). Testy projektuje się tak, aby umożliwiały sprawdzenie wszystkich zdefiniowanych zachowań (tj. zachowania podstawowego, wyjątkowego — zwanego też alternatywnym — oraz obsługę błędów). Pokrycie można mierzyć jako iloraz liczby przetestowanych zachowań określonych przez przypadki użycia przez łączną liczbę zachowań określonych przez przypadki użycia, zwykle wyrażony w procentach.

Więcej informacji na temat kryteriów pokrycia dotyczących testowania opartego na przypadkach użycia można znaleźć w [ISTQB-CTAL-TA], a także w [Roman 2015].

4.3. Białoskrzynkowe techniki testowania

Testowanie białoskrzynkowe opiera się na strukturze wewnętrznej przedmiotu testów. Techniki testowania białoskrzynkowego mogą być stosowane na wszystkich poziomach testów, jednakże dwie techniki związane z kodem, które omówiono w tym podrozdziale, najczęściej stosuje się na poziomie testów modułowych. Istnieją również bardziej zaawansowane techniki, których używa się w systemach krytycznych ze względów bezpieczeństwa, w systemach o newralgicznym znaczeniu dla działalności przedsiębiorstwa oraz w środowiskach wymagających wysokiego poziomu integralności (w celu uzyskania pełniejszego pokrycia), ale nie są one przedmiotem tego opracowania.

Więcej informacji na temat tych technik można znaleźć w [ISTQB-CTAL-TTA] oraz w [Roman 2015].

4.3.1. Testowanie i pokrycie instrukcji kodu

Testowanie instrukcji służy do sprawdzania potencjalnie wykonywalnych instrukcji zawartych w kodzie. Pokrycie mierzy się procentowo jako iloraz liczby instrukcji wykonanych przez testy przez łączną liczbę instrukcji wykonywalnych w przedmiocie testów, zazwyczaj wyrażany w procentach.

4.3.2. Testowanie i pokrycie decyzji

Testowanie decyzji służy do sprawdzania decyzji zawartych w kodzie oraz kodu wykonywanego na podstawie wyników decyzji. W tym celu tworzy się przypadki testowe, które odzwierciedlają przepływy sterowania występujące po punkcie decyzyjnym. W przypadku instrukcji *IF* tworzy się jeden przypadek dotyczący spełnienia warunku i jeden przypadek dotyczący niespełnienia warunku, a w przypadku instrukcji *CASE* niezbędne są przypadki testowe odpowiadające wszystkim możliwym wynikom, włącznie z wynikiem domyślnym.

Pokrycie mierzy się jako iloraz liczby wyników decyzji wykonanych przez testy przez łączną liczbę możliwych wyników decyzji w przedmiocie testów, zwykle wyrażony w procentach.

4.3.3. Korzyści wynikające z testowania instrukcji i testowania decyzji

Uzyskanie stuprocentowego pokrycia instrukcji kodu gwarantuje, że wszystkie instrukcje wykonywalne zawarte w kodzie zostały przetestowane co najmniej raz, nie gwarantuje natomiast, że przetestowana została cała logika decyzyjna. Jeśli chodzi o techniki białoskrzynkowe omówione w tym sylabusie, testowanie instrukcji kodu może zapewnić mniejsze pokrycie niż testowanie decyzji.

Uzyskanie stuprocentowego pokrycia decyzji oznacza, że wykonano wszystkie wyniki decyzji, czyli przetestowano zarówno wyniki „prawda”, jak i wyniki „fałsz” — nawet jeśli instrukcja warunkowa nie zawiera bloku instrukcji odpowiadającego decyzji „fałsz" (np. IF bez bloku ELSE). Pokrycie instrukcji kodu pomaga znaleźć defekty w kodzie, który nie został przetestowany przy użyciu innych testów, a pokrycie decyzji — w kodzie, w przypadku którego w innych testach nie uwzględniono wszystkich możliwych wyników decyzji.

Uzyskanie stuprocentowego pokrycia decyzji gwarantuje stuprocentowe pokrycie instrukcji kodu (ale nie odwrotnie).

4.4. Techniki testowania oparte na doświadczeniu

W ramach technik opartych na doświadczeniu przypadki testowe projektuje się z wykorzystaniem umiejętności i intuicji testerów oraz ich doświadczenia z podobnymi aplikacjami i technologiami. Techniki te pomagają w identyfikowaniu przypadków testowych, które trudno jest zidentyfikować przy użyciu innych, bardziej usystematyzowanych technik. Z drugiej strony uzyskany poziom pokrycia i skuteczność mogą bardzo różnić się w zależności od podejścia i doświadczenia testera — w pewnych przypadkach pokrycie może być trudne do oszacowania, a nawet niemożliwe do zmierzenia.

W kolejnych punktach omówiono najczęściej stosowane techniki oparte na doświadczeniu.

4.4.1. Zgadywanie błędów

Zgadywanie błędów[6] to technika pozwalająca przewidywać wystąpienie błędów, defektów i awarii na podstawie wiedzy testera dotyczącej między innymi:

- dotychczasowego działania aplikacji;
- typowych popełnianych pomyłek;
- awarii, które wystąpiły w innych aplikacjach.

Metodyczne podejście do zgadywania błędów polega na stworzeniu listy potencjalnych pomyłek, defektów i awarii, a następnie zaprojektowaniu testów pozwalających uwidocznić te awarie i znaleźć pomyłki, które je spowodowały. Listy pomyłek, defektów i awarii można opracowywać na podstawie własnego doświadczenia, danych dotyczących defektów i awarii oraz powszechnej wiedzy na temat przyczyn awarii oprogramowania.

[6] Formalnie rzecz ujmując, trzymając się terminologii ISTQB®, ten typ techniki powinien nazywać się „zgadywaniem defektów", ale pozostawiamy nazwę „zgadywanie błędów" ze względu na historyczne uwarunkowania (przyp. tłum.).

4.4.2. Testowanie eksploracyjne

Testowanie eksploracyjne polega na projektowaniu, wykonywaniu i rejestrowaniu testów nieformalnych (tj. testów, które nie zostały wcześniej zaprojektowane) oraz dokonywaniu ich oceny w sposób dynamiczny podczas ich wykonywania. Rezultaty testów dostarczają wiedzy na temat modułu lub systemu, a także pomagają tworzyć testy dotyczące obszarów wymagających dalszego przetestowania.

Testowanie eksploracyjne jest czasami przeprowadzane w formie tzw. testowania w sesjach, na potrzeby ustrukturyzowania aktywności. W testowaniu w sesjach, testowanie eksploracyjne odbywa się w ściśle określonym przedziale czasu, a tester prowadzi testy zgodnie z kartą sesji testowej (zawierającą cele testu) do przeprowadzenia testu. Co więcej, tester może udokumentować wykonane kroki i uzyskane informacje w karcie sesji testowej.

Testowanie eksploracyjne jest najbardziej przydatne w przypadku niepełnych lub niewłaściwie sporządzonych specyfikacji bądź w przypadku testowania pod presją czasu. Ponadto może być uzupełnieniem innych, bardziej formalnych technik testowania.

Testowanie eksploracyjne jest mocno związane z reaktywną strategią testowania (zobacz p. 5.2.2.). W ramach testowania eksploracyjnego można korzystać także z innych technik (czarnoskrzynkowych, białoskrzynkowych i opartych na doświadczeniu).

4.4.3. Testowanie w oparciu o listę kontrolną

W testowaniu w oparciu o listę kontrolną testerzy projektują, implementują i uruchamiają testy tak, aby pokryć warunki testowe wymienione w liście kontrolnej. Testerzy tworzą nowe listy kontrolne lub rozszerzają istniejące, ale mogą również korzystać z gotowych list kontrolnych bez ich modyfikacji. Listy kontrolne można opracowywać na podstawie własnego doświadczenia, danych dotyczących potencjalnych defektów i awarii, znajomości oczekiwań użytkowników lub wiedzy na temat przyczyn i objawów awarii oprogramowania.

Listy kontrolne można tworzyć na potrzeby różnych typów testów, w tym na potrzeby testów funkcjonalnych i niefunkcjonalnych. W przypadku braku szczegółowych przypadków testowych, testowanie w oparciu o listę kontrolną zapewnia niezbędne wytyczne i pozwala uzyskać pewien stopień spójności procesu testowania. Listy mają charakter wysokopoziomowy, w związku z czym podczas faktycznego testowania może występować pewna zmienność przekładająca się na większe pokrycie, ale kosztem mniejszej powtarzalności.

5. Zarządzanie testami — 225 min.

Słowa kluczowe

kierownik testów, kryteria wejścia, kryteria wyjścia, monitorowanie testów i nadzór nad testami, plan testów, planowanie testów, podejście do testowania, poziom ryzyka, raport o defekcie, raport o postępie testów, ryzyko, ryzyko produktowe, ryzyko projektowe, strategia testów, sumaryczny raport z testów, szacowanie testów, testowanie oparte na ryzyku, tester, zarządzanie defektami, zarządzanie konfiguracją

Cele nauczania — zarządzanie testami

5.1. Organizacja testów

FL-5.1.1. (K2) Kandydat potrafi omówić zalety i wady niezależnego testowania.

FL-5.1.2. (K1) Kandydat potrafi wskazać zadania kierownika testów i testera.

5.2. Planowanie i szacowanie testów

FL-5.2.1. (K2) Kandydat potrafi scharakteryzować cel i treść planu testów.

FL-5.2.2. (K2) Kandydat potrafi rozróżnić poszczególne strategie testów.

FL-5.2.3. (K2) Kandydat potrafi podać przykłady potencjalnych kryteriów wejścia i wyjścia.

FL-5.2.4. (K3) Kandydat potrafi wykorzystać wiedzę na temat ustalania priorytetów oraz zależności technicznych i logicznych do zaplanowania wykonania określonego zbioru przypadków testowych.

FL-5.2.5. (K1) Kandydat potrafi wskazać czynniki wpływające na pracochłonność testowania.

FL-5.2.6. (K2) Kandydat potrafi wyjaśnić różnicę między dwiema technikami szacowania: techniką opartą na miarach i techniką ekspercką.

5.3. Monitorowanie testów i nadzór nad testami

FL-5.3.1. (K1) Kandydat pamięta miary stosowane w odniesieniu do testowania.

FL-5.3.2. (K2) Kandydat potrafi podsumować cele i treść raportów z testów oraz wskazać ich odbiorców.

5.4. Zarządzanie konfiguracją

FL-5.4.1. (K2) Kandydat potrafi podsumować, w jaki sposób zarządzanie konfiguracją wspomaga testowanie.

5.5. Czynniki ryzyka a testowanie

FL-5.5.1. (K1) Kandydat potrafi określić poziom ryzyka na podstawie prawdopodobieństwa i wpływu.

FL-5.5.2. (K2) Kandydat potrafi rozróżnić czynniki ryzyka projektowego i produktowego.

FL-5.5.3. (K2) Kandydat potrafi opisać na przykładach potencjalny wpływ analizy ryzyka produktowego na staranność i zakres testowania.

5.6. Zarządzanie defektami

FL-5.6.1. (K3) Kandydat potrafi sporządzać raporty o defektach zawierające informacje na temat defektów wykrytych podczas testowania.

5.1. Organizacja testów

5.1.1. Niezależne testowanie

Zadania związane z testowaniem mogą wykonywać zarówno osoby pełniące konkretne role w procesie testowym, jak i osoby pełniące inne role (np. klienci). Z jednej strony pewien stopień niezależności często zwiększa skuteczność wykrywania defektów, ponieważ działania autora i testera mogą być obarczone różnymi błędami poznawczymi (patrz podrozdział 1.5.). Z drugiej strony niezależność nie zastępuje znajomości produktu, a deweloperzy znający doskonale produkt mogą efektywnie wykrywać wiele defektów w tworzonym przez siebie kodzie.

Poniżej przedstawiono niektóre poziomy niezależności testowania (w kolejności od najniższego do najwyższego):

- brak niezależnych testerów (jest to jedyna forma testowania dostępna w sytuacji, w której programiści testują własny kod);
- niezależni deweloperzy lub testerzy pracujący w zespole deweloperskim lub projektowym (w tym przypadku deweloperzy mogą np. testować produkty tworzone przez współpracowników);
- niezależny zespół testowy działający w ramach organizacji, podlegający kierownictwu projektu lub dyrekcji;
- niezależni testerzy będący przedstawicielami działów biznesowych lub społeczności użytkowników oraz testerzy specjalizujący się w określonych typach testów, takich jak: testowanie użyteczności, testowanie zabezpieczeń, testowanie wydajnościowe, testowanie zgodności czy testowanie przenaszalności;
- niezależni testerzy spoza organizacji pracujący u klienta (ang. *in-house*) lub poza siedzibą firmy (ang. *outsourcing*).

W większości typów projektów zazwyczaj najlepiej sprawdza się uwzględnienie wielu poziomów testowania i powierzenie wykonywania części testów niezależnym testerom. W testowaniu (zwłaszcza na niższych poziomach testów, np. testowaniu modułowym) powinni również uczestniczyć programiści, ponieważ w ten sposób mogą kontrolować jakość swojej pracy.

Sposób zapewnienia niezależności testowania zależy od wybranego modelu cyklu życia oprogramowania. W przypadku zwinnego wytwarzania oprogramowania testerzy mogą zostać przydzieleni do pracy w ramach zespołu deweloperów, a w niektórych organizacjach stosujących metody zwinne mogą być uznawani za członków szerszego niezależnego zespołu testowego. Ponadto w takich organizacjach właściciele produktu mogą wykonywać na zakończenie każdej iteracji testowanie akceptacyjne mające na celu walidację historyjek użytkownika.

Wśród potencjalnych korzyści wynikających z niezależności testowania należy wymienić:

- prawdopodobieństwo wykrycia przez niezależnych testerów innego rodzaju awarii niż te wykryte przez deweloperów ze względu na różne doświadczenia, techniczne punkty widzenia i błędy poznawcze;
- możliwość zweryfikowania, zakwestionowania lub obalenia przez niezależnych testerów założeń przyjętych przez interesariuszy na etapie specyfikowania i implementowania systemu;
- niezależni testerzy po stronie dostawcy mogą w rzetelny i obiektywny sposób informować o testowanym systemie bez (politycznego) nacisku ze strony firmy, która ich zatrudniła.

Potencjalne wady niezależności testowania to między innymi:

- izolacja testerów od zespołu deweloperów może prowadzić do braku komunikacji, opóźnień w dostarczaniu informacji zwrotnych zespołowi wytwórczemu i złych relacji z zespołem wytwórczym;
- niebezpieczeństwo utraty przez programistów poczucia odpowiedzialności za jakość;
- niebezpieczeństwo potraktowania niezależnych testerów jako wąskiego gardła;
- ryzyko, że niezależni testerzy nie będą dysponowali ważnymi informacjami (np. na temat przedmiotu testów).

Wiele organizacji jest w stanie osiągnąć korzyści z niezależnego testowania jednocześnie unikając wpisanych w nie wad.

5.1.2. Zadania kierownika testów i testera

W niniejszym sylabusie omówiono dwie role związane z testowaniem — kierownika testów i testera. Czynności i zadania wykonywane przez osoby pełniące obie role zależą od kontekstu projektu i produktu, umiejętności konkretnych osób oraz specyfiki organizacji.

Kierownik testów ponosi ogólną odpowiedzialność za proces testowy i sprawne kierowanie czynnościami związanymi z testowaniem. Rolę tę może pełnić zarówno osoba zajmująca się zawodowo kierowaniem testami, jak i kierownik projektu, kierownik zespołu deweloperskiego lub kierownik ds. zapewnienia jakości. Ponadto w przypadku większych projektów lub organizacji kierownik lub koordynator testów może mieć pod sobą kilka zespołów testowych, którymi kierują liderzy testów lub testerzy prowadzący.

W zakres obowiązków kierownika testów wchodzą zwykle następujące zadania:

- opracowywanie lub dokonywanie przeglądu strategii testów i polityki testów danej organizacji;
- planowanie testów, w tym uwzględnienie kontekstu oraz zapoznanie się z celami testów i czynnikami ryzyka w zakresie testowania (zadanie to może obejmować dokonywanie wyboru podejść do testowania, szacowanie czasu, pracochłonności i kosztów testowania, pozyskiwanie zasobów, definiowanie poziomów testów i cykli testowych oraz planowanie zarządzania defektami);
- sporządzanie i aktualizowanie planu testów;
- koordynowanie realizacji strategii testów i planu testów wraz z kierownikami projektu i innymi interesariuszami;
- prezentowanie punktu widzenia testerów w ramach innych czynności projektowych (takich jak planowanie integracji);
- inicjowanie procesów analizy, projektowania, implementacji i wykonywania testów, monitorowanie rezultatów testów oraz sprawdzanie statusu kryteriów wyjścia (definicji ukończenia) oraz nadzór nad czynnościami związanymi z zakończeniem testów;
- przygotowanie i dostarczenie raportu z postępu testów i raportu sumarycznego z testów na podstawie zgromadzonych informacji;

- dostosowywanie planu testów do rezultatów i postępu testów (dokumentowanych niekiedy w raportach o postępie testów i sumarycznych raportach z ukończenia testów) oraz podejmowanie niezbędnych działań w zakresie nadzoru nad testami;
- udzielanie pomocy w tworzeniu efektywnego mechanizmu zarządzania konfiguracją testaliów i systemu zarządzania defektami;
- wprowadzanie odpowiednich miar służących do mierzenia postępu testów oraz oceniania jakości testowania i produktu;
- udzielanie pomocy przy wyborze i implementacji narzędzi służących do realizacji procesu testowego, w tym przy określaniu budżetu przeznaczonego na wybór narzędzi (i potencjalnie na ich zakup lub pomoc techniczną) oraz wyznaczanie czasu na realizację projektów pilotażowych, a także udzielanie dalszej pomocy w zakresie korzystania z narzędzi;
- podejmowanie decyzji o implementacji środowisk testowych;
- monitorowanie procesu testowania oraz sporządzanie i przedstawianie raportów z testów na podstawie zgromadzonych informacji;
- promowanie testerów i zespołu testowego oraz reprezentowanie ich punktu widzenia w organizacji;
- stwarzanie możliwości rozwijania umiejętności i kariery testerów (np. poprzez plan szkoleń, ewaluacje osiągnięć, coaching).

Sposób wykonywania obowiązków kierownika testów zależy od cyklu życia oprogramowania. W ramach zwinnego wytwarzania oprogramowania niektóre z wyżej wymienionych zadań, a zwłaszcza zadania związane z codziennym testowaniem w obrębie zespołu zwinnego, wykonuje tester należący do tego zespołu. Niektóre zadania, które obejmują swoim zasięgiem kilka zespołów lub całą organizację bądź są związane z zarządzaniem kadrami, mogą wykonywać kierownicy testów spoza zespołu tworzącego oprogramowaie (zwani niekiedy trenerami testowymi). Zarządzanie procesem testowym opisane jest bardziej szczegółowo w [Black 2009].

Typowe zadania testera to między innymi:

- dokonywanie przeglądu planów testów i uczestniczenie w ich opracowywaniu;
- analizowanie, dokonywanie przeglądu i ocenianie wymagań, historyjek użytkownika i kryteriów akceptacji, specyfikacji oraz modeli (tj. podstawy testów) pod kątem ich testowalności;
- identyfikowanie i dokumentowanie warunków testowych oraz rejestrowanie powiązań między przypadkami testowymi, warunkami testowymi a podstawą testów;
- projektowanie, konfigurowanie i weryfikowanie środowisk testowych (często w porozumieniu z administratorami systemu i sieci);
- projektowanie i implementowanie przypadków testowych i skryptów testowych;
- przygotowywanie i pozyskiwanie danych testowych;
- tworzenie szczegółowego harmonogramu wykonywania testów;
- wykonywanie testów, ocenianie rezultatów i dokumentowanie odchyleń od oczekiwanych rezultatów;

- korzystanie z odpowiednich narzędzi usprawniających proces testowy;
- automatyzowanie testowania w zależności od potrzeb (zadanie to może być wykonywane z pomocą programisty lub eksperta ds. automatyzacji testowania);
- ewaluacja charakterystyk niefunkcjonalnych, takich jak: wydajność, niezawodność, użyteczność, zabezpieczenia, zgodność, przenaszalność;
- dokonywanie przeglądu testów opracowanych przez inne osoby.

Osoby zajmujące się analizą testów, projektowaniem testów, tworzeniem określonych typów testów czy automatyzacją testowania mogą być specjalistami w tych dziedzinach. Ponadto, w zależności od czynników ryzyka związanych z produktem i projektem oraz od wybranego modelu cyklu życia oprogramowania, na różnych poziomach testowania rolę testerów mogą przyjmować różne osoby. Na poziomie testowania modułowego i integracji modułów testerami są często programiści, na poziomie testowania akceptacyjnego — analitycy biznesowi, eksperci merytoryczni i użytkownicy, na poziomie testowania systemowego i integracji systemów — członkowie niezależnego zespołu testowego, a na poziomie produkcyjnych testów akceptacyjnych — operatorzy i/lub administratorzy systemu.

5.2. Planowanie i szacowanie testów

5.2.1. Cel i treść planu testów

Plan testów przedstawia w sposób skrótowy czynności testowe wykonywane w ramach projektów wytwarzania i pielęgnacji oprogramowania. Na proces planowania wpływają: polityka i strategia testów obowiązujące w danej organizacji, stosowane cykle życia i metody wytwarzania oprogramowania (patrz podrozdział 2.1.), zakres testowania, cele, czynniki ryzyka, ograniczenia, krytyczność, testowalność oraz dostępność zasobów. Ponadto, w miarę realizacji projektu i planu testów, udostępniane są dodatkowe informacje, które pozwalają uwzględnić w planie testów więcej szczegółów.

Planowanie testów to czynność o charakterze ciągłym, która jest wykonywana przez cały cykl życia oprogramowania (przy czym cykl ten może wykraczać poza zakres projektu i obejmować również fazę pielęgnacji). Ważnym jej elementem jest wykorzystanie informacji zwrotnych z czynności testowych do rozpoznawania zmieniających się czynników ryzyka i odpowiedniego korygowania planów. Rezultaty procesu planowania mogą zostać udokumentowane w głównym planie testów oraz w oddzielnych planach dotyczących poszczególnych poziomów testów (takich jak testowanie systemowe i akceptacyjne) bądź typów testów (takich jak testowanie użyteczności czy testowanie wydajnościowe). Planowanie testów może obejmować następujące działania (przy czym niektóre z nich mogą zostać udokumentowane w planie testów):

- określanie zakresu i celów testowania oraz związanego z nim ryzyka;
- określanie ogólnego podejścia do testowania;
- integrowanie i koordynowanie czynności testowych w ramach czynności związanych z cyklem życia oprogramowania;
- podejmowanie decyzji dotyczących tego, co należy przetestować, jakie kadrowe i inne zasoby będą niezbędne do wykonania poszczególnych czynności testowych oraz w jaki sposób należy wykonać te czynności;
- planowanie czynności związanych z analizą, projektowaniem, implementacją, wykonywaniem i oceną testów poprzez określenie konkretnych terminów (np. w ramach sekwencyjnego

wytwarzania oprogramowania) lub umieszczanie tych czynności w kontekście poszczególnych iteracji (np. w przypadku iteracyjnego wytwarzania oprogramowania);

- dokonywanie wyboru miar służących do monitorowania i nadzorowania testów;
- określanie budżetu potrzebnego na czynności testowe;
- ustalanie poziomu szczegółowości i struktury dokumentacji testów (np. poprzez udostępnianie szablonów lub przykładowych dokumentów).

Zawartość planu testów może się zmieniać, może też być szersza od zakresu opisanego powyżej. Przykładowy wzorzec planu testów i przykładowy plan testów zostały opisane w standardzie ISO [ISO/IEC/IEEE 29119-3].

5.2.2. Strategie testów i podejście do testowania

Strategia testów jest dokumentem wysokopoziomowym i zapewnia ogólny opis realizowanego procesu testowego, zazwyczaj na poziomie produktu bądź organizacji. Do typowych strategii testów zalicza się:

- **Strategia analityczna.** Strategia ta opiera się na analizie określonego czynnika (np. wymagań lub ryzyka). Przykładem podejścia analitycznego jest testowanie oparte na ryzyku, w którym punktem wyjścia do projektowania testów i ustalania ich priorytetów jest poziom ryzyka.
- **Strategia oparta na modelu.** W przypadku tej strategii testy projektuje się na podstawie modelu konkretnego wymaganego aspektu produktu, np.: funkcji, procesu biznesowego, struktury wewnętrznej bądź charakterystyki niefunkcjonalnej (takiej jak niezawodność). Przykłady takich modeli to: modele procesów biznesowych, modele stanów czy modele wzrostu niezawodności.
- **Strategia metodyczna.** Podstawą tej strategii jest systematyczne stosowanie z góry określonego zbioru testów lub warunków testowych, takich jak listy kontrolne zawierające typowe lub prawdopodobne typy awarii, listy ważnych charakterystyk jakościowych czy obowiązujące w firmie standardy wyglądu i działania (ang. *look-and-feel*) aplikacji mobilnych i stron internetowych.
- **Strategia zgodna z procesem** (lub standardem). Strategia ta polega na analizie, projektowaniu i implementacji przypadków testowych na podstawie zewnętrznych reguł i standardów (wynikających na przykład z norm branżowych), dokumentacji procesów bądź rygorystycznego identyfikowania i wykorzystania podstawy testów, a także na podstawie wszelkich procesów lub standardów narzuconych organizacji lub przez organizację.
- **Strategia kierowana** (lub konsultatywna). Podstawą tej strategii są przede wszystkim porady, wskazówki i instrukcje interesariuszy, ekspertów merytorycznych lub ekspertów technicznych — również spoza zespołu testowego lub organizacji.
- **Strategia testowa minimalizująca regresję.** Ta strategia – motywowana chęcią uniknięcia regresji już istniejących funkcjonalności – przewiduje ponowne wykorzystanie dotychczasowych testaliów (zwłaszcza przypadków testowych), szeroką automatyzację testów regresji oraz stosowanie standardowych zestawów testowych.
- **Strategia reaktywna.** W przypadku tej strategii testowanie jest ukierunkowane bardziej na reagowanie na zdarzenia niż na realizację ustalonego z góry planu (jak w przypadku wyżej opisanych strategii), a testy są projektowane i mogą być natychmiast wykonywane w oparciu o wiedzę uzyskaną dzięki rezultatom wcześniejszych testów.

Właściwa strategia testowa zazwyczaj jest kombinacją powyższych strategii. Przykładem takiej sytuacji jest testowanie oparte na ryzyku (strategia analityczna), które może być połączone z testowaniem eksploracyjnym (strategia reaktywna), ponieważ te strategie uzupełniają się nawzajem i wpływają na efektywność testowania, gdy są stosowane jednocześnie.

O ile strategia testów oferuje ogólny opis procesu testowego, o tyle podejście do testowania dostosowuje ją do potrzeb konkretnego projektu lub konkretnej wersji oprogramowania/systemu. Podejście do testowania jest punktem wyjścia do dokonania wyboru technik testowania, poziomów testów i typów testów oraz zdefiniowania kryteriów wejścia i wyjścia (lub definicji gotowości i definicji ukończenia). Dostosowując podejście do konkretnej sytuacji, należy podjąć określone decyzje wynikające ze złożoności i celów projektu, typu wytwarzanego produktu oraz analizy ryzyka produktowego. Wybrane podejście zależy od kontekstu i może uwzględniać takie czynniki jak: ryzyko, bezpieczeństwo, dostępność zasobów i umiejętności, technologie, charakter systemu (np. system wytworzony na zamówienie albo oprogramowanie do powszechnej sprzedaży), cele testów oraz obowiązujące przepisy.

5.2.3. Kryteria wejścia i wyjścia (definicja gotowości i definicja ukończenia)

W celu zapewnienia skutecznego nadzoru nad jakością oprogramowania i testowania zaleca się określenie kryteriów wskazujących, kiedy należy rozpocząć daną czynność testową oraz kiedy można ją uznać za zakończoną. Kryteria wejścia (w przypadku zwinnego wytwarzania oprogramowania zwane zwykle definicją gotowości) określają warunki wstępne, które muszą zostać spełnione przed rozpoczęciem danej czynności testowej. W przypadku niespełnienia kryteriów wejścia wykonanie tej czynności może być trudniejsze oraz bardziej czasochłonne, kosztowne i ryzykowne. Kryteria wyjścia (w przypadku zwinnego wytwarzania oprogramowania zwane zwykle definicją ukończenia) określają warunki, które muszą zostać spełnione, aby można było uznać wykonywanie testów danego poziomu lub zbioru testów za zakończone. Kryteria wejścia i wyjścia należy zdefiniować w odniesieniu do każdego poziomu i typu testów (kryteria te różnią się w zależności od celów testów).

Typowe kryteria wejścia to między innymi:

- dostępność testowalnych wymagań, historyjek użytkownika i/lub modeli (np. w przypadku stosowania strategii testowej opartej na modelu);
- dostępność elementów testowych, które spełniły kryteria wyjścia obowiązujące na wcześniejszych poziomach testów;
- dostępność środowiska testowego;
- dostępność niezbędnych narzędzi testowych;
- dostępność danych testowych i innych niezbędnych zasobów.

Typowe kryteria wyjścia to między innymi:

- zakończenie wykonywania zaplanowanych testów;
- osiągnięcie odpowiedniego poziomu pokrycia (tj. pokrycia wymagań, historyjek użytkownika i kryteriów akceptacji oraz kodu);
- nieprzekroczenie uzgodnionego limitu nieusuniętych defektów;
- uzyskanie wystarczająco niskiej, szacowanej gęstości defektów,

- uzyskanie wystarczająco wysokich wskaźników niezawodności, wydajności, użyteczności, zabezpieczeń i innych istotnych charakterystyk.

Nawet w przypadku niespełnienia kryteriów wyjścia, czynności testowe mogą zostać skrócone z powodu wykorzystania całego budżetu, upływu zaplanowanego czasu i/lub presji związanej z wprowadzeniem produktu na rynek. Zakończenie testowania w takich okolicznościach może być dopuszczalne, o ile interesariusze i właściciele biznesowi projektu znają i akceptują ryzyko związane z uruchomieniem systemu bez dalszego testowania.

5.2.4. Harmonogram wykonywania testów

Po opracowaniu poszczególnych przypadków i procedur testowych (oraz ewentualnym zautomatyzowaniu niektórych z nich) oraz zgrupowaniu tychże w zestawy testowe, można utworzyć na ich podstawie harmonogram wykonywania testów, który określa kolejność ich uruchamiania. Harmonogram ten powinien uwzględniać takie czynniki jak: priorytety, zależności, konieczność wykonania testów potwierdzających i regresji oraz najbardziej efektywną kolejność wykonywania testów.

O ile jest to możliwe, testy powinny być wykonywane w kolejności odpowiadającej poziomom priorytetów (zwykle najpierw wykonuje się testy o najwyższym priorytecie). Praktyka ta może jednak nie mieć zastosowania, jeśli przypadki testowe lub testowane przez nie cechy są uzależnione od innych elementów. Przykładem może być sytuacja, gdy przypadek testowy o wyższym priorytecie jest uzależniony od przypadku o niższym priorytecie — należy wtedy w pierwszej kolejności wykonać przypadek o niższym priorytecie. Analogicznie: jeśli występują wzajemne zależności między kilkoma przypadkami testowymi, kolejność ich wykonywania należy ustalić niezależnie od priorytetów. Ponadto może zajść konieczność priorytetowego wykonania testów potwierdzających i regresji w zależności od tego, jak ważne jest szybkie uzyskanie informacji zwrotnych na temat zmian, przy czym w takim przypadku może być również konieczne uwzględnienie ewentualnych współzależności.

W niektórych przypadkach możliwe są różne sekwencje testów różniące się efektywnością. W takiej sytuacji należy odpowiednio wyważyć kwestie efektywności wykonywania testów oraz przestrzegania ustalonych priorytetów.

5.2.5. Czynniki wpływające na pracochłonność testowania

Jednym z elementów zarządzania testami jest szacowanie ich pracochłonności, czyli przewidywanie nakładów pracy związanych z testowaniem, które są niezbędne do osiągnięcia celów testowania w przypadku danego projektu bądź danej wersji lub iteracji. Do czynników wpływających na pracochłonność testowania należą między innymi: charakterystyka produktu, charakterystyka procesu wytwarzania oprogramowania, czynniki ludzkie oraz rezultaty testów.

Charakterystyka produktu

- czynniki ryzyka związane z produktem;
- jakość specyfikacji (tj. podstawy testów);
- wielkość produktu;
- złożoność dziedziny produktu;
- wymagania dotyczące charakterystyk jakościowych (np. bezpieczeństwa i niezawodności);
- wymagany poziom szczegółowości dokumentacji testów;

- wymagania dotyczące zgodności z przepisami prawa.

Charakterystyka procesu wytwarzania oprogramowania

- stabilność i dojrzałość organizacji;
- stosowany model wytwarzania oprogramowania (np. model kaskadowy, model zwinny);
- podejście do testowania;
- używane narzędzia;
- proces testowy;
- presja czasu.

Czynniki ludzkie

- umiejętności i doświadczenie zaangażowanych osób, zwłaszcza doświadczenie z podobnymi projektami/produktami (np. doświadczenie branżowe);
- umiejętność współpracy [7]zespołu i umiejętności kierownika.

Rezultaty testów

- liczba i ważność wykrytych defektów;
- liczba wymaganych poprawek.

5.2.6. Techniki szacowania testów

Do oszacowania nakładów pracy niezbędnych do prawidłowego wykonania testów w ramach projektu można użyć różnych technik. Dwie najczęściej stosowane techniki to:

- technika oparta na miarach — szacowanie pracochłonności testowania na podstawie miar z wcześniejszych podobnych projektów lub na podstawie wartości typowych;
- technika ekspercka — szacowanie pracochłonności testowania na podstawie doświadczenia osób odpowiedzialnych za zadania związane z testowaniem lub przez ekspertów.

Przykładem zastosowania podejścia opartego na miarach w kontekście zwinnego wytwarzania oprogramowania są wykresy spalania (ang. *burndown charts*). Rejestrowane i raportowane przy ich użyciu nakłady pracy są następnie wykorzystywane do określania prędkości zespołu (ang. *team velocity*), czyli ilości pracy, jaką zespół ten może wykonać w następnej iteracji. Z kolei poker planistyczny to przykład podejścia eksperckiego, ponieważ członkowie zespołu szacują nakłady pracy niezbędne do dostarczenia danej funkcjonalności na podstawie własnego doświadczenia (patrz [ISTQB-CTFL-AT]).

Przykładem zastosowania podejścia opartego na miarach w projektach sekwencyjnych są modele usuwania defektów, które przewidują rejestrowanie i raportowanie liczby defektów oraz czasu niezbędnego do ich usunięcia, a następnie szacowanie na tej podstawie pracochłonności przyszłych projektów o podobnym charakterze. Z kolei szerokopasmowa technika delficka jest przykładem

[7] W wersji angielskiej użyto sformułowania „*team cohesion*” – wydaje się, że określenie „umiejętność współpracy” dobrze oddaje tę ideę.

podejścia eksperckiego, zgodnie z którym grupy ekspertów przedstawiają wartości szacunkowe na podstawie dotychczasowego doświadczenia (patrz [ISTQB-CTAL-TM]).

5.3. Monitorowanie testów i nadzór nad testami

Celem monitorowania testów jest gromadzenie i udostępnianie informacji pozwalających uzyskać wgląd w przebieg czynności testowych. Informacje objęte monitorowaniem, które mogą być zbierane ręcznie lub automatycznie, powinny być wykorzystywane do oceny postępu testów oraz pomiaru spełnienia kryteriów wyjścia dotyczących poszczególnych poziomów testów lub wykonania zadań testowych związanych z definicją ukończenia w projektach zwinnych (takich jak osiągnięcie zakładanego pokrycia czynników ryzyka produktowego, wymagań lub kryteriów akceptacji).

Nadzór nad testami obejmuje wszelkie działania zarządcze i korygujące podejmowane na podstawie zgromadzonych, wykazanych w raportach informacji i miar. Powyższe działania mogą dotyczyć dowolnych czynności testowych i wpływać na pozostałe czynności związane z cyklem życia oprogramowania.

Działania wykonywane w ramach nadzoru nad testami to między innymi:

- ponowne ustalanie priorytetów testów w przypadku zmaterializowania się zidentyfikowanego czynnika ryzyka (np. nieterminowego dostarczenia oprogramowania);
- wprowadzanie zmian w harmonogramie testów zależnie od dostępności lub niedostępności środowiska testowego lub innych zasobów;
- dokonywanie ponownej oceny kryteriów wejścia lub wyjścia w związku z wprowadzeniem poprawek.

5.3.1. Miary stosowane w testowaniu

Podczas wykonywania testów danego poziomu i po ich zakończeniu można zbierać miary pozwalające oszacować:

- postęp realizacji harmonogramu i budżetu;
- bieżącą jakość przedmiotu testów;
- adekwatność wybranego podejścia do testowania;
- skuteczność czynności testowych z punktu widzenia realizacji celów.

Powszechnie stosowane są następujące miary dotyczące testów:

- procent wykonania zaplanowanych prac związanych z przygotowaniem przypadków testowych (lub procent przygotowania zaplanowanych przypadków testowych);
- procent wykonania zaplanowanych prac związanych z przygotowaniem środowiska testowego;
- poziom wykonania przypadków testowych (np. liczba wykonanych/niewykonanych przypadków testowych, zaliczonych/niezaliczonych przypadków testowych i/lub zaliczonych/niezaliczonych warunków testowych);
- informacje o defektach (np. gęstość defektów, liczba wykrytych i usuniętych defektów, współczynnik awarii oraz rezultaty testów potwierdzających);

- pokrycie testowe wymagań, historyjek użytkownika i kryteriów akceptacji, czynników ryzyka lub kodu;
- koszty testowania, w tym porównanie tych kosztów z korzyściami wynikającymi z wykrycia następnego defektu lub wykonania następnego testu.

5.3.2. Cel, treść i odbiorcy raportów z testów

Raporty z testów służą do podsumowywania i przekazywania informacji na temat czynności testowych (np. testów danego poziomu) zarówno w trakcie ich wykonywania, jak i po ich zakończeniu. Raport z testów sporządzany podczas wykonywania czynności testowej może nosić nazwę raportu o postępie testów, a raport sporządzany w momencie zakończenia takiej czynności — sumarycznego raportu z testów.

Na etapie monitorowania i nadzoru nad testami kierownik testów regularnie sporządza raporty o postępie testów dla interesariuszy. Oprócz części wspólnych, dla raportu z postępu testów, sumarycznego raportu z testów, typowy raport z postępu testów może zawierać informacje dotyczące:

- statusu czynności testowych i postępu realizacji planu testów;
- czynników zakłócających wykonywanie prac;
- testów zaplanowanych w następnym okresie raportowania;
- jakości przedmiotu testów.

Po spełnieniu kryteriów wyjścia kierownik testów sporządza sumaryczny raport z testów. Dokument ten zawiera podsumowanie wykonanych testów zgodnie z najnowszym raportem z postępu testów oraz innych istotnych informacji.

Typowy sumaryczny raport z testów zawiera między innymi:

- podsumowanie wykonanych testów;
- informacje na temat zdarzeń, które miały miejsce w okresie testowania;
- informacje o odstępstwach od planu, włączając w to odstępstwa od harmonogramu, czasu trwania lub wysiłku związanego z testowaniem;
- informacje o statusie testowania i jakości produktu z punktu widzenia kryteriów wyjścia (lub definicji ukończenia);
- informacje o czynnikach, które blokowały lub nadal blokują przebieg testów;
- miary związane z defektami, przypadkami testowymi, pokryciem testowym, postępem prac oraz wykorzystaniem zasobów (np. miary opisane w p. 5.3.1.);
- informacje na temat ryzyka (resztkowego);
- informacje o wytworzonych produktach pracy związanych z testowaniem, które mogą zostać ponownie wykorzystane.

Treść raportu z testów zależy od projektu, wymagań organizacyjnych i cyklu życia oprogramowania. I tak przykładem może być złożony projekt z wieloma interesariuszami lub projekt podlegający określonym regulacjom prawnym, co może wymagać bardziej szczegółowego i rygorystycznego raportowania niż szybka aktualizacja oprogramowania. Ponadto w przypadku zwinnego wytwarzania oprogramowania raportowanie o postępie testów można zintegrować z tablicami zadań, przeglądami

defektów i wykresami spalania, które są omawiane podczas codziennych spotkań roboczych (ang. *daily stand-up meetings*) (patrz [ISTQB-CTAL-AT]).

Raporty z testów muszą być dostosowane zarówno do kontekstu projektu, jak i do potrzeb docelowych odbiorców. Typ i ilość informacji umieszczonych w raporcie przeznaczonym dla odbiorców technicznych lub zespołu testowego mogą być inne od tych zamieszczonych w raporcie sumarycznym dla kierownictwa. W pierwszym przypadku ważne mogą być szczegółowe informacje na temat typów defektów i związanych z nimi trendów, w drugim zaś bardziej odpowiedni może być raport wysokiego poziomu (zawierający np. podsumowanie statusu defektów według priorytetów, budżetu i harmonogramu oraz zaliczonych, niezaliczonych i niewykonanych przypadków testowych).

Standard ISO [ISO/IEC/IEEE 29119-3] odwołuje się do dwóch typów raportów, tj. raportu o postępie testów i raportu z ukończenia testów (nazywanego w tym sylabusie sumarycznym raportem z testów), a także zawiera struktury i przykłady dotyczące każdego z tych typów.

5.4. Zarządzanie konfiguracją

Celem zarządzania konfiguracją jest zapewnienie i utrzymanie integralności modułu/ lub systemu i testaliów oraz wzajemnych relacji między nimi przez cały cykl życia projektu i oprogramowania.

W kontekście testowania zarządzanie konfiguracją może polegać na zagwarantowaniu tego, aby:

- wszystkie przedmioty testów zostały zidentyfikowane, objęte kontrolą wersji i śledzeniem zmian oraz powiązane ze sobą wzajemnie;
- wszystkie testalia zostały zidentyfikowane, objęte kontrolą wersji i śledzeniem zmian oraz powiązane ze sobą wzajemnie i z wersjami przedmiotu testów w sposób pozwalający utrzymać możliwość śledzenia na wszystkich etapach procesu testowego;
- wszystkie zidentyfikowane dokumenty i elementy oprogramowania były przywoływane w sposób jednoznaczny w dokumentacji testów.

Procedury zarządzania konfiguracją wraz z niezbędną infrastrukturą (narzędziami) należy zidentyfikować i zaimplementować na etapie planowania testów.

5.5. Czynniki ryzyka a testowanie

5.5.1. Definicja ryzyka

Ryzyko jest możliwością wystąpienia w przyszłości zdarzenia o niepożądanych konsekwencjach. O poziomie ryzyka decyduje prawdopodobieństwo wystąpienia niekorzystnego zdarzenia oraz jego wpływ (tj. wynikające z niego szkody).

5.5.2. Czynniki ryzyka produktowego i projektowego

Ryzyko produktowe występuje wszędzie tam, gdzie produkt pracy (np. specyfikacja, moduł, system lub test) może nie zaspokoić uzasadnionych potrzeb użytkowników i/lub interesariuszy. Czynniki ryzyka produktowego związane z konkretnymi charakterystykami jakościowymi produktu (np. przydatność funkcjonalna, niezawodność, wydajność, użyteczność, utrzymywalność, przenaszalność) są również nazywane czynnikami ryzyka jakościowego.

Przykładami czynników ryzyka produktowego mogą być następujące problemy:

- niewykonywanie przez oprogramowanie zakładanych funkcji zgodnie ze specyfikacją;
- niewykonywanie zakładanych funkcji oprogramowania zgodnie z potrzebami użytkowników, klientów i/lub interesariuszy;
- niedostateczne spełnienie przez architekturę systemu określonych wymagań niefunkcjonalnych;
- niepoprawne wykonywanie konkretnych obliczeń w niektórych okolicznościach;
- błędy w kodzie struktury sterowania pętlą;
- zbyt długi czas odpowiedzi w systemie transakcyjnym wysokiej wydajności;
- niezgodność informacji zwrotnych na temat doświadczenia użytkownika z oczekiwaniami dotyczącymi produktu.

Ryzyko projektowe obejmuje sytuacje, których zaistnienie może mieć negatywny wpływ na możliwość osiągnięcia celów projektu. Przykłady ryzyka projektowego to:

- problemy związane z projektem, takie jak:
  - potencjalne opóźnienia w dostawach, ukończeniu zadań bądź spełnieniu kryteriów wyjścia (definicji ukończenia);
  - niedokładne oszacowanie, realokacja środków do projektów o wyższym priorytecie lub ogólne cięcia kosztów w całej organizacji, które mogą skutkować niewystarczającym finansowaniem projektu;
  - wprowadzenie w ostatniej chwili zmian wymagających dokonania licznych przeróbek;
- problemy organizacyjne, takie jak:
  - niewystarczające kwalifikacje lub przeszkolenie pracowników bądź braki kadrowe;
  - konflikty i problemy wynikające z doboru personelu;
  - brak dostępności użytkowników, pracowników struktur biznesowych lub ekspertów merytorycznych z powodu sprzecznych priorytetów biznesowych;
- problemy natury politycznej, takie jak:
  - niewłaściwe przekazywanie przez testerów informacji na temat ich potrzeb i/lub rezultatów testów;
  - niepodjęcie przez programistów/testerów dalszych działań na podstawie informacji uzyskanych w wyniku testowania i przeglądów (np. niewprowadzenie udoskonaleń w procedurach wytwarzania i testowania oprogramowania);
  - niewłaściwe podejście do testowania lub oczekiwania związane z testowaniem (np. niedocenianie korzyści wynikających z wykrycia defektów podczas testowania);
- problemy techniczne, takie jak:
  - niedostateczne doprecyzowanie wymagań;

- brak możliwości spełnienia wymagań z uwagi na ograniczenia czasowe;
- nieudostępnienie na czas środowiska testowego;
- zbyt późne przeprowadzenie konwersji danych, zaplanowanie migracji lub udostępnienie potrzebnych do tego narzędzi;
- wady w procesie wytwarzania oprogramowania mogące wpływać na spójność lub jakość produktów prac projektowych, kodu, konfiguracji, danych testowych i przypadków testowych;
- kumulacja defektów lub innego rodzaju dług techniczny powstały na skutek problemów z zarządzaniem defektami lub innych podobnych problemów;

- problemy związane z dostawcami, takie jak:
  - niedostarczenie niezbędnego produktu lub usługi przez zewnętrznego dostawcę bądź ogłoszenie przez niego upadłości;
  - utrudnienia w realizacji projektu wynikające z problemów związanych z umowami.

Ryzyko projektowe może dotyczyć zarówno czynności związanych z wytwarzaniem oprogramowania, jak i jego testowaniem. W niektórych przypadkach za zarządzanie ryzykiem projektowym odpowiadają kierownicy projektów, ale zdarza się również, że za czynniki ryzyka projektowego związane z testowaniem odpowiedzialność ponoszą kierownicy testów.

5.5.3. Testowanie oparte na ryzyku a jakość produktu

Wiedzę na temat ryzyka można wykorzystać do odpowiedniego ukierunkowania działań wykonywanych podczas testowania. Na podstawie ryzyka można podejmować decyzje co do tego, gdzie i kiedy należy rozpocząć testowanie, a także identyfikować obszary wymagające większej uwagi. Celem testowania jest zmniejszenie prawdopodobieństwa wystąpienia niekorzystnego zdarzenia bądź ograniczenie jego wpływu. W związku z tym testowanie przyczynia się do łagodzenia ryzyka, a także dostarcza informacje na temat zidentyfikowanych czynników ryzyka — w tym ryzyka resztkowego (rezydualnego[8], nie rozwiązanego).

Podejście do testowania oparte na ryzyku umożliwia prewencyjne obniżanie poziomu ryzyka produktowego. Jednym z elementów tego podejścia jest analiza ryzyka produktowego, która obejmuje identyfikowanie czynników tego typu ryzyka oraz szacowanie prawdopodobieństwa wystąpienia i wpływu każdego z nich. Uzyskane w ten sposób informacje na temat ryzyka produktowego można wykorzystać do planowania testów, specyfikowania, przygotowywania i wykonywania przypadków testowych oraz monitorowania i nadzorowania testów. Wczesna analiza ryzyka produktowego przyczynia się do powodzenia całego projektu.

W przypadku podejścia opartego na ryzyku wynik przeprowadzonej analizy ryzyka produktowego umożliwia:

- wskazanie odpowiednich technik testowania;
- określenie konkretnych typów testów, które należy wykonać (np. testy zabezpieczeń, testy dostępności);

[8] ryzyko resztkowe (rezydualne) to ryzyko pozostające w testowanym obiekcie po podjęciu wszelkich możliwych bądź też wszelkich (ekonomicznie zasadnych) kroków zmierzających do jego uniknięcia (przyp. tłum.).

- określenie zakresu wykonywanych testów;
- ustalenie priorytetów testowania w sposób sprzyjający jak najwcześniejszemu wykryciu defektów krytycznych;
- ustalenie, czy w celu zmniejszenia ryzyka należy wykonać inne czynności niezwiązane z testowaniem (np. przeprowadzić szkolenie dla niedoświadczonych projektantów).

Testowanie oparte na ryzyku pozwala skorzystać ze wspólnej wiedzy i spostrzeżeń interesariuszy projektu do przeprowadzenia analizy ryzyka produktowego. Aby zminimalizować prawdopodobieństwo awarii produktu, w ramach zarządzania ryzykiem wykonuje się usystematyzowane czynności obejmujące:

- analizowanie potencjalnych problemów (czynników ryzyka) i regularne dokonywanie ich ponownej oceny;
- ustalanie, które czynniki ryzyka są istotne i wymagają podjęcia działań;
- podejmowanie działań mających na celu złagodzenie ryzyka;
- tworzenie planów awaryjnych na wypadek faktycznego wystąpienia określonych czynników ryzyka.

Ponadto testowanie pozwala zidentyfikować nowe czynniki ryzyka, wskazać czynniki wymagające złagodzenia oraz zmniejszyć niepewność związaną z ryzykiem.

5.6. Zarządzanie defektami

Jednym z celów testowania jest znajdowanie defektów, w związku z czym wszystkie znalezione defekty należy zalogować. Sposób zgłoszenia defektu zależy od kontekstu testowania danego modułu lub systemu, poziomu testów oraz wybranego modelu wytwarzania oprogramowania. Każdy zidentyfikowany defekt powinien zostać zbadany i objęty śledzeniem od momentu wykrycia i sklasyfikowania, do momentu rozwiązania problemu (tj. usunięcia defektu i pomyślnego zakończenia testów potwierdzających, odroczenia do kolejnej wersji, zaakceptowania defektu jako trwałego ograniczenia produktu itd.). Aby umożliwić zarządzanie wszystkimi defektami do czasu ich usunięcia, organizacja powinna wdrożyć proces zarządzania defektami (obejmujący również odpowiedni przepływ pracy i reguły klasyfikacji defektów). Proces ten musi zostać uzgodniony ze wszystkimi podmiotami uczestniczącymi w zarządzaniu defektami, w tym z architektami, projektantami, deweloperami, testerami oraz właścicielami produktu. W niektórych organizacjach proces zgłaszania i śledzenia defektów może być nieformalny.

W procesie zarządzania defektami może się okazać, że w niektórych raportach opisane zostały rezultaty fałszywie pozytywne, a nie rzeczywiste awarie spowodowane defektami. Przykładem takiej sytuacji może być niezaliczenie testu na skutek przerwania połączenia sieciowego lub przekroczenia limitu czasu. Zachowanie to nie jest skutkiem defektu w przedmiocie testów, ale należy je potraktować jako nieprawidłowość i zbadać. W związku z tym testerzy powinni starać się ograniczać do minimum liczbę rezultatów fałszywie pozytywnych, które są zgłaszane jako defekty.

Defekty można zgłaszać na etapie kodowania, analizy statycznej, przeglądów, podczas testowania dynamicznego lub podczas eksploatacji produktu. Raporty o defektach mogą dotyczyć problemów występujących w kodzie lub działającym systemie bądź we wszelkiego rodzaju dokumentacji, w tym w wymaganiach, historyjkach użytkownika i kryteriach akceptacji, dokumentach związanych z programowaniem lub testowaniem, podręcznikach użytkownika bądź podręcznikach instalacji.

Aby stworzyć skuteczny i efektywny proces zarządzania defektami, organizacje mogą określać własne standardy dotyczące atrybutów i klasyfikacji defektów oraz związanego z ich obsługą przepływu pracy.

Typowy raport o defekcie ma na celu:

- dostarczenie deweloperom i innym interesariuszom informacji na temat wszelkich zaistniałych zdarzeń w zakresie niezbędnym do zidentyfikowania skutku, wyizolowania problemu i umożliwienie skorygowania ewentualnych defektów;
- umożliwienie kierownikom testów śledzenia jakości produktu pracy i wpływu na postęp testowania (np. znalezienie dużej liczby defektów będzie wymagało czasu na ich zgłoszenie, a następnie przeprowadzenie testów potwierdzających);
- przedstawienie sugestii dotyczących usprawnienia procesów wytwarzania i testowego.

Raport o defekcie przedkładany podczas testowania dynamicznego zawiera zwykle następujące, szczegółowe informacje:

- identyfikator;
- tytuł i krótkie podsumowanie zgłaszanego defektu;
- data raportu o defekcie, zgłaszająca jednostka organizacyjna i autor zgłoszenia;
- identyfikacja elementu testowego (testowanego elementu konfiguracji) i środowiska;
- faza cyklu życia oprogramowania, w której zaobserwowano defekt;
- opis defektu umożliwiający jego odtworzenie i usunięcie (w tym ewentualne dzienniki, zrzuty bazy danych, zrzuty ekranu lub nagrania), gdy defekt został wykryty podczas wykonywania testów;
- oczekiwane i rzeczywiste rezultaty;
- zakres lub stopień wpływu (ważność) defektu z punktu widzenia interesariuszy;
- priorytet usunięcia defektu;
- status raportu o defekcie (np. otwarty, odroczony, powielony, oczekujący na poprawkę, oczekujący na testowanie potwierdzające, ponownie otwarty, zamknięty);
- wnioski, zalecenia i zgody;
- kwestie globalne takie jak wpływ zmiany wynikającej z defektu na inne obszary oprogramowania/systemu;
- historia zmian, w tym sekwencja działań podjętych przez członków zespołu projektowego w celu zlokalizowania, usunięcia i potwierdzenia usunięcia defektu;
- odwołania do innych elementów, w tym do przypadku testowego, dzięki któremu problem został ujawniony.

Niektóre z powyższych szczegółowych informacji mogą zostać automatycznie uwzględnione i/lub objęte zarządzaniem w przypadku korzystania z narzędzi do zarządzania defektami (przykładem może być automatyczne przypisanie identyfikatora, przypisanie i aktualizacja stanu raportu o defekcie w ramach przepływu pracy itd.). Defekty wykryte podczas testowania statycznego (a zwłaszcza w trakcie

przeglądów) są zwykle dokumentowane w inny sposób, np. w notatkach ze spotkania związanego z przeglądem.

Przykładową treść raportu o defekcie przedstawiono w standardzie ISO [ISO/IEC/IEEE 29119-3] (raporty o defektach są w nim nazywane raportami o incydentach).

6. Narzędzia wspomagające testowanie — 40 min.

Słowa kluczowe

automatyzacja testowania, narzędzie do wykonywania testów, narzędzie do zarządzania testami, testowanie oparte na słowach kluczowych, testowanie sterowane danymi

Cele nauczania — narzędzia wspomagające testowanie

6.1. Uwarunkowania związane z narzędziami testowymi

FL-6.1.1. (K2) Kandydat potrafi sklasyfikować narzędzia testowe według przeznaczenia i obsługiwanych czynności testowych.

FL-6.1.2. (K1) Kandydat potrafi zidentyfikować korzyści i czynniki ryzyka związane z automatyzacją testowania.

FL-6.1.3. (K1) Kandydat pamięta o szczególnych uwarunkowaniach związanych z narzędziami do wykonywania testów i zarządzania testami.

6.2. Skuteczne korzystanie z narzędzi

FL-6.2.1. (K1) Kandydat potrafi wskazać główne zasady wyboru narzędzi.

FL-6.2.2. (K1) Kandydat pamięta cele stosowania projektów pilotażowych wprowadzających narzędzia.

FL-6.2.3. (K1) Kandydat potrafi wskazać czynniki sukcesu związane z oceną, implementacją, wdrożeniem i bieżącą obsługą narzędzi testowych w organizacji.

6.1. Uwarunkowania związane z narzędziami testowymi

Narzędzia testowe służą do wspierania jednej lub wielu czynności testowych. Dostępne są między innymi następujące narzędzia:

- narzędzia używane bezpośrednio podczas testowania, takie jak narzędzia do wykonywania testów i narzędzia do przygotowywania danych testowych;
- narzędzia ułatwiające zarządzanie wymaganiami, przypadkami testowymi, procedurami testowymi, skryptami testów automatycznych, rezultatami testów, danymi testowymi, defektami oraz narzędzia raportujące i monitorujące wykonywanie testów;
- narzędzia służące do analizy i oceny;
- inne narzędzia wspomagające testowanie (w tym znaczeniu narzędziem testowym jest również np. arkusz kalkulacyjny).

6.1.1. Klasyfikacja narzędzi testowych

Zależnie od kontekstu narzędzia testowe mogą służyć realizacji jednego lub kilku z poniżej wymienionych celów:

- zwiększenie efektywności wykonywania czynności testowych poprzez zautomatyzowanie powtarzających się zadań lub tych zadań, których ręczne wykonywanie wymaga dużych zasobów lub istotnego wysiłku (np. wykonania testów, testowania regresji);
- podniesienie wydajności czynności testowych poprzez wspieranie czynności manualnych w trakcie całego procesu testowego (patrz podrozdział 1.4.);
- podniesienie jakości wykonywanych czynności testowych poprzez zwiększenie spójności testowania i odtwarzalności defektów;
- zautomatyzowanie czynności, których nie można wykonać ręcznie (np. testowania wydajnościowego na dużą skalę);
- zwiększenie niezawodności testowania (np. poprzez zautomatyzowanie porównań obejmujących duże ilości danych bądź symulowanie zachowań).

Narzędzia mogą być klasyfikowane według kilku kryteriów, takich jak: przeznaczenie, cena, model licencjonowania (np. komercyjne lub *open source*) oraz stosowana technologia. W tym sylabusie narzędzia sklasyfikowano na podstawie wspomaganych przez nie czynności testowych.

Niektóre narzędzia obsługują tylko lub głównie jedną czynność, natomiast inne pozwalają wykonywać kilka czynności. W drugim przypadku podstawą klasyfikacji jest czynność, z którą dane narzędzie jest najściślej związane. Należy również pamiętać, że narzędzia oferowane przez jednego dostawcę, a zwłaszcza narzędzia zaprojektowane z myślą o współpracy ze sobą, mogą być dostarczane w postaci zintegrowanego pakietu.

Niektóre typy narzędzi testowych mogą mieć charakter inwazyjny, co oznacza, że mogą wpływać na rzeczywisty wynik testu. Przykładem są dodatkowe instrukcje wykonywane przez narzędzie do testów wydajnościowych, które mogą powodować wydłużenie czasu odpowiedzi aplikacji, a zastosowanie narzędzia mierzącego pokrycie kodu może zakłócić pomiar pokrycia. Zjawisko to jest nazywane efektem próbnika.

Niektóre narzędzia posiadają funkcje, które są bardziej odpowiednie dla deweloperów. Przykładem mogą być narzędzia używane podczas testowania modułowego i integracyjnego. W kolejnych punktach narzędzia te oznaczono literą „D".

Narzędzia wspomagające zarządzanie testowaniem i testaliami

Narzędzia do zarządzania mogą obejmować swoim działaniem wszystkie czynności testowe wykonywane przez cały cykl życia oprogramowania. Do narzędzi wspomagających zarządzanie testowaniem i testaliami zaliczyć można między innymi:

- narzędzia do zarządzania testami i narzędzia do zarządzania cyklem życia aplikacji (ang. *Application Lifecycle Management tools* ALM);
- narzędzia do zarządzania wymaganiami (np. śledzenie powiązań z obiektami testów);
- narzędzia do zarządzania defektami;
- narzędzia do zarządzania konfiguracją;
- narzędzia do ciągłej integracji (D).

Narzędzia wspomagające testowanie statyczne

Narzędzia do testowania statycznego są związane z czynnościami i korzyściami opisanymi w rozdziale 3. Do narzędzi tych zaliczamy między innymi:

- narzędzia do analizy statycznej (D).

Narzędzia wspomagające projektowanie i implementację testów

Narzędzia do projektowania testów ułatwiają tworzenie utrzymywalnych produktów pracy — w tym przypadków testowych, procedur testowych i danych testowych — na etapie projektowania i implementacji testów. Do narzędzi tych zaliczamy między innymi:

- narzędzia do testowania opartego na modelu;
- narzędzia do przygotowywania danych testowych.

W niektórych przypadkach narzędzia wspomagające projektowanie i implementację testów mogą również wspomagać ich wykonywanie i logowanie (bądź przekazywać dane wyjściowe do innych narzędzi wspomagających wykonywanie i logowanie testów).

Narzędzia wspomagające wykonywanie i logowanie testów

Istnieje wiele narzędzi wspomagających i usprawniających wykonywanie i logowanie testów. Do narzędzi tych zaliczamy między innymi:

- narzędzia do wykonywania testów (np. do uruchamiania testów regresji);
- narzędzia mierzące pokrycie (np. pokrycie wymagań lub kodu) (D);
- jarzma testowe (D).

Narzędzia wspomagające pomiar wydajności i analizę dynamiczną

Narzędzia do pomiaru wydajności i analizy dynamicznej są niezbędne do wykonywania czynności związanych z testowaniem wydajnościowym i obciążeniowym, ponieważ czynności tych nie da się skutecznie wykonywać ręcznie. Do narzędzi tych zaliczamy między innymi:

- narzędzia do testów wydajnościowych;
- narzędzia do analizy dynamicznej (D).

Narzędzia wspomagające wyspecjalizowane czynności testowe

Oprócz narzędzi wspomagających ogólny proces testowy istnieje również wiele innych narzędzi, które są pomocne w bardziej szczegółowym testowaniu charakterystyk niefunkcjonalnych.

6.1.2. Korzyści i czynniki ryzyka związane z automatyzacją testowania

Sam zakup narzędzia nie gwarantuje jeszcze sukcesu. Osiągnięcie realnych i trwałych korzyści z wdrożenia w organizacji nowego narzędzia zawsze wymaga dodatkowego wysiłku. Ponadto, obok potencjalnych korzyści i szans z użytkowania narzędzi do testowania, wiąże się z nimi również określone ryzyko, zwłaszcza w przypadku narzędzi do wykonywania testów (np. w projektach automatyzacji testowania).

Potencjalne korzyści wynikające z zastosowania narzędzi wspomagających testowanie to między innymi:

- ograniczenie powtarzalnych czynności wykonywanych ręcznie (takich jak uruchamianie testów regresji, wykonywanie zadań związanych z konfigurowaniem i de-konfigurowaniem środowiska, wielokrotne wprowadzanie tych samych danych testowych czy sprawdzanie zgodności ze standardami tworzenia kodu), co pozwala na zaoszczędzenie czasu;
- zwiększenie spójności i powtarzalności (np. poprzez spójne tworzenie danych testowych, wykonywanie testów przy użyciu narzędzia w tej samej kolejności i z tą samą częstotliwością bądź wyprowadzanie testów w spójny sposób z wymagań);
- bardziej obiektywna ocena rezultatów testów (np. w zakresie miar statycznych czy pokrycia);
- łatwiejszy dostęp do informacji na temat testowania (np. do danych statystycznych i wykresów obrazujących postęp testów, współczynników występowania defektów oraz danych dotyczących wydajności).

Do potencjalnych czynników ryzyka związanych z zastosowaniem narzędzi wspomagających testowanie należą:

- nierealistyczne oczekiwania wobec narzędzia (dotyczące między innymi funkcjonalności i łatwości obsługi);
- niedoszacowanie czaso- i pracochłonności oraz kosztów początkowego wdrożenia narzędzia (w tym szkoleń i korzystania z usług zewnętrznych ekspertów);
- niedoszacowanie czaso- i pracochłonności działań niezbędnych do osiągnięcia znaczących i trwałych korzyści z tytułu używania narzędzia (w tym nakładów niezbędnych do wprowadzenia zmian w procesie testowym oraz ciągłego doskonalenia sposobu wykorzystania narzędzia);
- niedoszacowanie nakładów pracy związanych z utrzymaniem produktów pracy generowanych przez narzędzie;

- nadmierne uzależnienie od narzędzia (np. traktowanie narzędzia jako substytutu projektowania lub wykonywania testów bądź stosowanie automatycznego testowania w sytuacji, w której lepsze byłoby testowanie ręczne);
- zaniedbanie kontroli wersji produktów pracy;
- zlekceważenie zależności pomiędzy newralgicznymi narzędziami (takimi jak: narzędzia do zarządzania wymaganiami, narzędzia do zarządzania konfiguracją, narzędzia do zarządzania defektami czy narzędzia od różnych dostawców) a problemami związanymi z ich współdziałaniem;
- możliwość zakończenia przez dostawcę działalności, wycofania narzędzia lub sprzedaży narzędzia innemu dostawcy;
- niska jakość usług dostawcy w zakresie pomocy technicznej bądź dostarczania uaktualnień lub poprawek usuwających defekty w narzędziu;
- możliwość wstrzymania realizacji projektu w przypadku narzędzia typu *open source*;
- możliwość braku współpracy narzędzia z nowymi platformami lub technologiami;
- możliwość braku jednoznacznego określenia odpowiedzialności za narzędzie (np. w zakresie mentoringu, aktualizacji itd.).

6.1.3. Szczególne uwarunkowania związane z narzędziami do wykonywania testów i zarządzania testami

Aby sprawnie i pomyślnie zaimplementować narzędzia do wykonywania testów i zarządzania testami w organizacji, na etapie ich wyboru i integracji należy uwzględnić kilka istotnych czynników.

Narzędzia do wykonywania testów
Narzędzia do wykonywania testów umożliwiają uruchamianie przedmiotu testów przy użyciu skryptów testów automatycznych. W przypadku tego typu narzędzi uzyskanie istotnych korzyści wymaga często dużego nakładu pracy.

- **Rejestrowanie działań wykonywanych ręcznie**: projektowanie testów poprzez rejestrowanie działań wykonywanych ręcznie przez testera wydaje się atrakcyjnym rozwiązaniem, ale podejście to nie zapewnia skalowalności niezbędnej w przypadku dużej liczby skryptów testowych. Zarejestrowany skrypt jest liniowym odwzorowaniem obejmującym konkretne dane i akcje, w związku z czym może okazać się niestabilny w przypadku wystąpienia nieprzewidzianych zdarzeń. Wygenerowane skrypty nadal wymagają ciągłej pielęgnacji w związku z rozwojem interfejsu użytkownika systemu.
- **Podejście do testowania sterowanego danymi**: to podejście zakłada wydzielenie danych wejściowych i oczekiwanych rezultatów (zwykle do arkusza kalkulacyjnego) oraz zastosowanie bardziej generycznego skryptu testowego, który odczytuje dane wejściowe i wykonuje testy przy użyciu różnych danych.
- **Testowanie oparte na słowach kluczowych**: w tym podejściu do testowania stosuje się generyczny skrypt, który przetwarza słowa kluczowe opisujące wykonywane akcje (zwane także słowami akcji), a następnie wywołuje odpowiednie skrypty przetwarzające dostarczone dane testowe.

Powyższe podejścia wymagają eksperckiej znajomości języka skryptowego (przez testerów, deweloperów lub specjalistów z dziedziny automatyzacji testowania), jakkolwiek w przypadku korzystania z testów sterowanych danymi lub opartych na słowach kluczowych testerzy, którzy nie znają

języka skryptowego, mogą również przyczynić się do utworzenia danych testowych i/lub słów kluczowych dla tych predefiniowanych skryptów. Bez względu na zastosowaną technikę skryptową, oczekiwane rezultaty poszczególnych testów należy porównać z rzeczywistymi rezultatami testu wykonanego metodą dynamiczną (porównanie podczas wykonywania testu) lub z wykorzystaniem zapisanych — do późniejszego wykorzystania — informacji (porównanie po wykonaniu).

Dalsze informacje i przykłady dotyczące podejścia sterowanego danymi i testowania opartego na słowach kluczowych można znaleźć w [ISTQB-CTAL-TAE], a także w [Fewster 1999] oraz [Buwalda 2001].

Narzędzia do testowania opartego na modelu pozwalają zarejestrować specyfikację funkcjonalną w postaci modelu takiego jak diagram aktywności. Zadanie to wykonuje zazwyczaj projektant systemu. Narzędzie interpretuje model i tworzy na jego podstawie specyfikacje przypadków testowych, które można następnie zapisać w narzędziu do zarządzania testami i/lub wykonać za pomocą narzędzia do wykonywania testów (patrz [ISTQB-CTAL-MBT]).

Narzędzia do zarządzania testami

Narzędzia do zarządzania testami muszą często komunikować się z innymi narzędziami lub arkuszami kalkulacyjnymi z wielu powodów, np.:

- w celu generowania użytecznych informacji w formacie zgodnym z potrzebami organizacji;
- w celu utrzymania możliwości śledzenia powiązań z wymaganiami w narzędziu do zarządzania wymaganiami;
- w celu tworzenia powiązań z informacjami o wersjach przedmiotu testów w narzędziu do zarządzania konfiguracją.

Kwestię tę należy wziąć pod uwagę zwłaszcza w przypadku korzystania z narzędzia zintegrowanego (np. narzędzia do zarządzania cyklem życia aplikacji), które obejmuje zarówno moduł zarządzania, jak i inne moduły (dostarczające np. informacje na temat harmonogramu i budżetu projektu), które są używane przez różne grupy użytkowników w organizacji.

6.2. Skuteczne korzystanie z narzędzi

6.2.1. Główne zasady wyboru narzędzi

Najważniejsze kwestie, które należy wziąć pod uwagę przy wyborze narzędzia dla organizacji, to między innymi:

- dokonanie oceny dojrzałości własnej organizacji oraz jej mocnych i słabych stron;
- wskazanie możliwości udoskonalania procesu testowego dzięki wykorzystaniu narzędzi;
- zapoznanie się z technologiami stosowanymi w ramach przedmiotu testów w celu wybrania zgodnego z nimi narzędzia;
- zrozumienie działania narzędzi do budowania wersji i ciągłej integracji, które są już używane w danej organizacji, w celu zapewnienia kompatybilności i integracji;
- dokonanie oceny narzędzia zgodnie z jednoznacznymi wymaganiami i obiektywnymi kryteriami;
- uwzględnienie dostępności (i czasu trwania) bezpłatnego okresu próbnego w przypadku danego narzędzia;
- dokonanie oceny dostawcy (między innymi w zakresie szkoleń, pomocy technicznej i aspektów handlowych) lub możliwości obsługi narzędzi niekomercyjnych (np. *open source*);
- zidentyfikowanie wewnętrznych wymagań dotyczących coachingu i mentoringu w zakresie korzystania z narzędzia;
- dokonanie oceny potrzeb szkoleniowych z uwzględnieniem umiejętności w zakresie testowania (i automatyzacji testowania) posiadanych przez osoby, które będą pracować bezpośrednio z danym narzędziem;
- uwzględnienie zalet i wad poszczególnych modeli licencyjnych (np. licencji komercyjnych lub *open source*);
- oszacowanie stosunku kosztów do korzyści na podstawie konkretnego przypadku biznesowego (jeśli jest to wymagane).

Ostatnim krokiem procesu wyboru narzędzia powinno być przeprowadzenie dowodu słuszności przyjętej koncepcji w celu ustalenia, czy dane narzędzie działa sprawnie w połączeniu z testowanym oprogramowaniem i w ramach bieżącej infrastruktury oraz wskazanie ewentualnych zmian w infrastrukturze, które będą niezbędne do skutecznego korzystania z wybranego narzędzia.

6.2.2. Projekty pilotażowe związane z wprowadzaniem narzędzia w organizacji

Po dokonaniu wyboru narzędzia i pomyślnym przeprowadzeniu dowodu słuszności przyjętej koncepcji (ang. *Proof of Concept - PoC)* można rozpocząć wprowadzanie tego narzędzia w organizacji. Pierwszym etapem tego procesu jest zwykle projekt pilotażowy, który ma na celu:

- szczegółowe zapoznanie się z narzędziem (w tym z jego mocnymi i słabymi stronami);
- ustalenie, na ile dane narzędzie jest dopasowane do istniejących procesów i praktyk oraz wskazanie niezbędnych zmian;
- podjęcie decyzji w zakresie standardowych sposobów użytkowania, przechowywania i utrzymywania narzędzia i testowych produktów pracy oraz zarządzania nimi (np. ustalenie

konwencji nazewnictwa plików i testów, wybór standardów tworzenia kodu, utworzenie bibliotek oraz określenie modułowości zestawów testowych);

- oszacowanie, czy spodziewane korzyści będzie można osiągnąć uzasadnionym kosztem;
- zapoznanie się z miarami, które mają być zbierane i raportowane przez narzędzie oraz skonfigurowanie narzędzia w sposób gwarantujący ich zbieranie i raportowanie.

6.2.3. Czynniki sukcesu związane z narzędziami

Wśród czynników sukcesu związanych z oceną, implementacją, wdrożeniem i bieżącą obsługą narzędzia w organizacji należy wymienić:

- przyrostowe wdrażanie narzędzia w pozostałych częściach organizacji;
- dostosowywanie i udoskonalanie procesów pod kątem współpracy z narzędziem;
- zapewnienie użytkownikom narzędzia odpowiednich szkoleń, coachingu i mentoringu;
- określenie wytycznych dotyczących korzystania z narzędzia (np. wewnętrznych standardów dotyczących automatyzacji);
- wprowadzenie mechanizmu zbierania informacji na temat sposobów użytkowania narzędzia;
- monitorowanie użytkowania narzędzia i wynikających z tego korzyści;
- zapewnienie użytkownikom narzędzia niezbędnej pomocy technicznej;
- zbieranie doświadczeń od wszystkich użytkowników.

Należy również zadbać o integrację techniczną i organizacyjną narzędzia w każdym cyklu życia oprogramowania, w którego realizację mogą być zaangażowane odrębne jednostki organizacyjne odpowiedzialne za działalność operacyjną i/lub zewnętrzni dostawcy.

Więcej informacji na temat użytkowania narzędzi do wykonywania testów można znaleźć np. w [Graham 2012].

7. Bibliografia

Normy i standardy

ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering — Software testing — Part 1: Concepts and definitions

ISO/IEC/IEEE 29119-2 (2013) Software and systems engineering — Software testing — Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2013) Software and systems engineering — Software testing — Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2015) Software and systems engineering — Software testing — Part 4: Test techniques

ISO/IEC 25010 (2011) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

ISO/IEC 20246 (2017) Software and systems engineering — Work product reviews

UML 2.5 — Unified Modeling Language Reference Manual (2017), http://www.omg.org/spec/UML/2.5.1/

Dokumenty ISTQB® [9]

ISTQB® S Słownik terminów testowych

ISTQB® Certyfikowany tester. Przegląd poziomu podstawowego 2018

ISTQB-CTFL-MBT Testowanie oparte na modelu. Sylabus rozszerzenia poziomu podstawowego (E)

ISTQB-CTFL-AT Tester zwinny. Sylabus rozszerzenia poziomu podstawowego

ISTQB-CTAL-ATA Analityk testów. Sylabus poziomu zaawansowanego

ISTQB-CTAL-TM Kierownik testów. Sylabus poziomu zaawansowanego

ISTQB-CTAL-TTA Techniczny analityk testów. Sylabus poziomu zaawansowanego

ISTQB-CTAL-SEC Testowanie zabezpieczeń. Sylabus poziomu zaawansowanego (E)

ISTQB-CTAL-TAE Inżynieria automatyzacji testowania. Sylabus poziomu zaawansowanego

ISTQB-CTEL-ETM Kierownik testów. Sylabus poziomu eksperckiego (E)

ISTQB-CTEL-EITP Doskonalenie procesu testowego. Sylabus poziomu eksperckiego.

[9] Dokumenty oznaczone symbolem (E) nie zostały przetłumaczone na język polski.

Książki i artykuły

Beizer, B. (1990*) Software Testing Techniques (2e)*, Van Nostrand Reinhold: Boston MA

Black, R. (2017) *Agile Testing Foundations*, BCS Learning & Development Ltd: Swindon UK

Black, R. (2009) *Managing the Testing Process (3e)*, John Wiley & Sons: New York NY

Buwalda, H. et al. (2001) *Integrated Test Design and Automation*, Addison Wesley: Reading MA

Copeland, L. (2004) *A Practitioner's Guide to Software Test Design*, Artech House: Norwood MA

Craig, R., Jaskiel, S. (2002) *Systematic Software Testing*, Artech House: Norwood MA

Crispin, L., Gregory, J. (2008) *Agile Testing*, Pearson Education: Boston MA

Fewster, M., Graham, D. (1999) *Software Test Automation*, Addison Wesley: Harlow UK

Gilb, T., Graham, D. (1993) *Software Inspection*, Addison Wesley: Reading MA

Graham, D., Fewster, M. (2012) *Experiences of Test Automation*, Pearson Education: Boston MA

Gregory, J., Crispin, L. (2015) *More Agile Testing*, Pearson Education: Boston MA

Jorgensen, P. (2014) *Software Testing, A Craftsman's Approach (4e)*, CRC Press: Boca Raton FL

Kaner, C., Bach, J., Pettichord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York NY

Kaner, C., Padmanabhan, S., Hoffman, D. (2013) *The Domain Testing Workbook*, Context-Driven Press: New York NY

Kramer, A., Legeard, B. (2016) *Model-Based Testing Essentials: Guide to the ISTQB Certified Model-Based Tester: Foundation Level*, John Wiley & Sons: New York NY

Myers, G. (2011) *The Art of Software Testing*, (3e), John Wiley & Sons: New York NY

Roman, A. (2015) *Testowanie i jakość oprogramowania. Modele, techniki, narzędzia,* PWN, Warszawa

Sauer, C. (2000) *The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated* Program of Research, IEEE Transactions on Software Engineering, Volume 26 (1)

Shull, F., Rus, I., Basili, V. (2000) How Perspective-Based Reading can Improve Requirement Inspections, *IEEE Computer, Volume 33, Issue 7, pp 73-79*

van Veenendaal, E. (ed.) (2004) *The Testing Practitioner* (Chapters 8 - 10), UTN Publishers: The Netherlands

Wiegers, K. (2002) *Peer Reviews in Software*, Pearson Education: Boston MA

Weinberg, G. (2008) *Perfect Software and Other Illusions about Testing*, Dorset House: New York NY

Inne pozycje (do których nie ma bezpośrednich odwołań w niniejszym sylabusie)

Black, R., van Veenendaal, E., Graham, D. (2019) *Foundations of Software Testing: ISTQB Certification (4)*, Cengage Learning: London, UK

Hetzel, W. (1993) *Complete Guide to Software Testing (2e)*, QED Information Sciences: Wellesley MA

Inne pozycje w języku polskim (do których nie ma bezpośrednich odwołań w niniejszym sylabusie)

Roman, A. (2016) *Statyczne testowanie oprogramowania,* Wydawnictwo Naukowe PWN, Warszawa

Roman, A., Zmitrowicz K. (2016) *Testowanie oprogramowania w praktyce,* część 1., Wydawnictwo Naukowe PWN, Warszawa

Roman, A., Zmitrowicz K. (2018) *Testowanie oprogramowania w praktyce*, część 2., Wydawnictwo Naukowe PWN, Warszawa

Smilgin, R. (2018), *Zawód tester. Od decyzji do zdobycia doświadczenia*, Wydawnictwo Naukowe PWN, Warszawa

Zmitrowicz, K. (2015) *Jakość projektów informatycznych. Rozwój i testowanie oprogramowania*, Wydawnictwo Helion.

8. Załącznik A. Kontekst sylabusa

Geneza dokumentu

Niniejszy dokument zawiera sylabus ISTQB® dla certyfikowanego testera na poziomie podstawowym, który stanowi pierwszy poziom międzynarodowego systemu kwalifikacji zatwierdzonego przez ISTQB® (www.istqb.org).

Dokument ten został przygotowany w okresie od czerwca do sierpnia 2019 r. przez grupę roboczą złożoną z członków powołanych przez International Software Testing Qualification Board (ISTQB®). Aktualizacje zostały dodane w oparciu o opinie członków Rad Krajowych, którzy korzystali z Sylabusa Poziomu Podstawowego 2018.

Poprzednia wersja Sylabusa Poziomu Podstawowego 2018 została opracowana w latach 2014–2018 przez grupę roboczą złożoną z osób wyznaczonych przez International Software Testing Qualifications Board (ISTQB®). Wersja z 2018 roku została wstępnie zweryfikowana przez przedstawicieli wszystkich Rad Krajowych ISTQB®, a następnie przez wybranych przedstawicieli międzynarodowej społeczności testerów oprogramowania.

Cele certyfikatu podstawowego

- Podkreślenie znaczenia testowania jako jednej z podstawowych specjalizacji zawodowych w ramach inżynierii oprogramowania.
- Stworzenie standardowych ram rozwoju zawodowego testerów.
- Stworzenie systemu umożliwiającego uznawanie kwalifikacji zawodowych testerów przez pracodawców, klientów i innych testerów oraz podnoszenie statusu testerów.
- Promowanie spójnych, dobrych praktyk testowania we wszystkich dyscyplinach inżynierii oprogramowania.
- Wskazanie zagadnień związanych z testowaniem, które są istotne i wartościowe dla całej branży.
- Stworzenie dostawcom oprogramowania możliwości zatrudniania certyfikowanych testerów i uzyskiwania przewagi konkurencyjnej nad innymi dostawcami poprzez reklamowanie przyjętej polityki rekrutacji testerów.
- Stworzenie testerom i osobom zainteresowanym testowaniem możliwości zdobycia uznawanych na całym świecie kwalifikacji w tej dziedzinie.

Cele międzynarodowego systemu uzyskiwania kwalifikacji

- Stworzenie podstaw do porównywania wiedzy w dziedzinie testowania w różnych krajach.
- Ułatwienie testerom znalezienia pracy w innych krajach.
- Zapewnienie wspólnego rozumienia zagadnień związanych z testowaniem w ramach projektów wielonarodowych i międzynarodowych.
- Zwiększenie liczby wykwalifikowanych testerów na całym świecie.
- Stworzenie inicjatywy o charakterze międzynarodowym, która zapewni większe korzyści i silniejsze oddziaływanie niż inicjatywy realizowane na szczeblu krajowym.

- Opracowanie wspólnego, międzynarodowego zbioru informacji i wiedzy w obszarze testowania w oparciu o sylabus i zawartą w nim terminologię oraz podnoszenie poziomu wiedzy na temat testowania wszystkich uczestników zaangażowanych w proces testowania.
- Promowanie zawodu testera w większej liczbie krajów.
- Umożliwienie testerom uzyskania powszechnie uznawanych kwalifikacji w języku ojczystym.
- Stworzenie testerom z różnych krajów warunków do dzielenia się wiedzą, zasobami i doświadczeniem.
- Zapewnienie międzynarodowego uznawania statusu testerów i niniejszej kwalifikacji z uwagi na udział przedstawicieli wielu krajów.

Wymagania stawiane kandydatom

Od osób przystępujących do egzaminu ISTQB® na certyfikowanego testera na poziomie podstawowym wymaga się zainteresowania tematyką testowania oprogramowania. Ponadto zdecydowanie zaleca się, aby kandydaci:

- posiadali co najmniej podstawowe doświadczenie w dziedzinie wytwarzania lub testowania oprogramowania, np. sześciomiesięczne doświadczenie na stanowisku testera wykonującego testy systemowe lub akceptacyjne bądź na stanowisku programisty;
- ukończyli kurs akredytowany przez jedną z Rad Krajowych ISTQB®, zgodnie ze standardami ISTQB®.

Okoliczności powstania i geneza powstania certyfikatu podstawowego w dziedzinie testowania oprogramowania

Niezależna certyfikacja testerów oprogramowania rozpoczęła się w 1998 roku w Wielkiej Brytanii. Pod auspicjami działającej w ramach British Computer Society Rady ds. Egzaminów Informatycznych (Information Systems Examination Board — ISEB) została wówczas powołana specjalna jednostka ds. testowania oprogramowania — Software Testing Board (www.bcs.org.uk/iseb). W 2002 roku własny system kwalifikacji testerów stworzyła również niemiecka organizacja ASQF (www.asqf.de). Niniejszy sylabus powstał na bazie sylabusów ISEB i ASQF, przy czym zawarte w nim informacje zostały przeorganizowane, zaktualizowane i uzupełnione. Główny nacisk położono na zagadnienia zapewniające testerom największe, praktyczne wsparcie.

Dotychczasowe certyfikaty podstawowe w dziedzinie testowania oprogramowania (np. certyfikaty wydawane przez ISEB, ASQF lub Rady Krajowe ISTQB®), które zostały przyznane przed wprowadzeniem certyfikatu międzynarodowego, będą uznawane za równoważne temu certyfikatowi. Certyfikat podstawowy nie wygasa i nie wymaga odnowienia. Data przyznania jest umieszczona na certyfikacie.

Uwarunkowania lokalne w poszczególnych krajach uczestniczących w programie leżą w gestii Rad Krajowych ISTQB®. Obowiązki Rad Krajowych określa ISTQB®, natomiast ich realizację pozostawiono poszczególnym organizacjom członkowskim. Do obowiązków Rad Krajowych należą zwykle akredytacja dostawców szkoleń i wyznaczanie terminów egzaminów.

9. Załącznik B. Cele nauczania i poziomy poznawcze

Poniżej przedstawiono cele nauczania obowiązujące dla niniejszego sylabusa. Znajomość każdego z zagadnień poruszonych w sylabusie będzie sprawdzana na egzaminie zgodnie z przypisanym celem nauczania.

Poziom 1 — zapamiętać (K1)

Kandydat rozpoznaje, pamięta lub umie sobie przypomnieć dany termin lub dane pojęcie.

Słowa kluczowe: zidentyfikować, zapamiętać, powtórzyć, przypomnieć sobie, rozpoznać, wiedzieć

Przykłady

Kandydat potrafi rozpoznać, że definicją „awarii" jest:

- „brak realizacji usługi na rzecz użytkownika lub dowolnego innego interesariusza" lub
- „odchylenie modułu lub systemu od oczekiwanego zachowania lub rezultatu działania".

Poziom 2 — zrozumieć (K2)

Kandydat potrafi uzasadnić lub wyjaśnić stwierdzenia dotyczące danego zagadnienia, a także podsumować, porównać i sklasyfikować pojęcia z zakresu testowania oraz podać odpowiednie przykłady.

Słowa kluczowe: podsumować, uogólnić, wyabstrahować, sklasyfikować, porównać, przypisać, przeciwstawić, podać przykład, zinterpretować, przetłumaczyć, przedstawić, wydedukować, wywnioskować, skategoryzować, skonstruować modele

Przykłady

Kandydat potrafi wyjaśnić, dlaczego analizę testów i projektowanie testów należy rozpocząć najwcześniej:

- Aby wykryć defekty w czasie, gdy ich usunięcie jest tańsze.
- Aby wykryć w pierwszej kolejności najważniejsze defekty.

Kandydat potrafi wyjaśnić podobieństwa i różnice między testowaniem integracyjnym a systemowym:

- Podobieństwa: przedmiotem testów w przypadku testowania integracyjnego i testowania systemowego może być kilka modułów; testowanie integracyjne i testowanie systemowe mogą obejmować testy niefunkcjonalne.
- Różnice: testowanie integracyjne koncentruje się na interfejsach i interakcjach, a testowanie systemowe — na aspektach działania systemu jako całości takich jak całościowe procesy biznesowe.

Poziom K3 — zastosować (K3)

Kandydat potrafi wybrać prawidłowe zastosowanie pojęcia lub techniki i zastosować je w danym kontekście.

Słowa kluczowe: zaimplementować, wykonać, użyć, postępować zgodnie z procedurą, zastosować procedurę

Przykłady

- Kandydat potrafi zidentyfikować wartości brzegowe dla poprawnych i niepoprawnych klas równoważności.
- Kandydat potrafi zaprojektować przypadki testowe na podstawie podanego diagramu przejść między stanami w sposób zapewniający pokrycie wszystkich przejść.

Materiały dodatkowe

w zakresie poziomów poznawczych celów nauczania

Anderson, L. W., Krathwohl, D. R. (red.) (2001) *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Allyn & Bacon, Boston MA.

10. Załącznik C. Opis wydania

Sylabus Poziomu Podstawowego ISTQB® 2018 V 3.1. to niewielka aktualizacja wersji 2018. Utworzono osobny opis wydania wersji 2018 V 3.1. z przeglądem poszczególnych rozdziałów. Dodatkowo została wydana wersja Sylabus Poziomu Podstawowego ISTQB® 2018 V 3.1. z zaznaczonymi zmianami.

Treść Sylabusa Poziomu Podstawowego ISTQB® w wersji 2018 została gruntownie zaktualizowana i przeredagowana w porównaniu z wersją z 2011 roku, w związku z tym nie sporządzono szczegółowego opisu wydania w podziale na rozdziały i podrozdziały, a jedynie przedstawiono podsumowanie najważniejszych zmian. Ponadto w oddzielnym dokumencie z opisem wydania przedstawiono powiązania pomiędzy celami nauczania sylabusa poziomu podstawowego w wersji 2011 a celami nauczania w wersji 2018 (ze wskazaniem elementów dodanych, zaktualizowanych lub usuniętych).

Zgodnie z danymi z początku 2017 roku do egzaminu na poziomie podstawowym przystąpiło ponad 550 tys. osób z ponad 100 krajów, a na całym świecie jest ponad 500 tys. certyfikowanych testerów. Przy założeniu, że wszystkie te osoby przeczytały sylabus poziomu podstawowego, aby zdać egzamin, dokument ten jest prawdopodobnie najczęściej czytanym opracowaniem dotyczącym testowania oprogramowania.

Obecna, gruntownie zaktualizowana wersja, czerpie z tego dziedzictwa, a jednocześnie pozwala jeszcze lepiej zaprezentować jakość usług, które ISTQB® oferuje globalnej społeczności testerów, kolejnym 500 tys. kandydatów.

W bieżącej wersji wszystkie cele nauczania zostały przeredagowane w taki sposób, aby każdy z nich stanowił niepodzielną całość oraz aby można było jednoznacznie powiązać każdy cel nauczania (i pytania egzaminacyjne) z rozdziałami/podrozdziałami, w których są one omawiane, a poszczególne zagadnienia — z celami nauczania, których dotyczą. Ponadto na nowo oszacowano (w porównaniu z wersją 2011) czas przeznaczony na naukę materiału zawartego w poszczególnych rozdziałach. W tym celu zastosowano sprawdzone mechanizmy heurystyczne i formuły używane w przypadku innych sylabusów ISTQB®, które działają w oparciu o analizę celów nauczania omawianych w każdym z rozdziałów.

Niniejszy sylabus poziomu podstawowego zawiera opis najlepszych praktyk i technik, które wytrzymały próbę czasu, ale dokonano w nim wielu zmian mających na celu bardziej nowoczesne przedstawienie materiału, zwłaszcza w zakresie modeli wytwarzania oprogramowania (takich jak Scrum czy ciągłe dostarczanie (ang. *continuous delivery*) oraz technologii (takich jak Internet rzeczy - IoT). Zaktualizowano również odwołania do norm i standardów, aby uwzględnić ich najnowsze wydania:

1. Standard ISO/IEC/IEEE 29119 zastąpił normę IEEE 829.
2. Standard ISO/IEC 25010 zastąpił normę ISO 9126.
3. Standard ISO/IEC 20246 zastąpił normę IEEE 1028.

Ponadto z uwagi na fakt, że oferta ISTQB® w ostatnich dziesięciu latach została znacznie rozszerzona, dodano liczne odwołania do materiałów pokrewnych zawartych w innych sylabusach ISTQB® oraz starannie zweryfikowano spójność ze wszystkimi sylabusami i słownikiem terminów testowych ISTQB® [ISTQB S]. Autorzy skupili się na możliwości praktycznego wykorzystania przedstawionych wiadomości oraz zachowaniu równowagi między wiedzą a umiejętnościami, dzięki czemu nowa wersja powinna być bardziej przystępna i zrozumiała oraz łatwiejsza do przetłumaczenia, a zawarty w niej materiał — łatwiejszy do opanowania.

Szczegółową analizę zmian dokonanych w Sylabusie Poziomu Podstawowego 2018 zawiera dokument „Certyfikowany tester poziomu podstawowego ISTQB - Informacje o wersji 2018 V 3.1.”.

”.

11. Indeks