Certyfikowany tester ISTQB®

Sylabus poziomu zaawansowanego Analityk testów (CTAL-TA)

wersja 3.1.2.4.

International Software Testing Qualifications Board®

© Stowarzyszenie Jakości Systemów Informatycznych

Informacja o prawach autorskich

Informacja o prawach autorskich © International Software Testing Qualifications Board (zwana dalej ISTQB®). ISTQB® jest zarejestrowanym znakiem handlowym International Software Testing Qualifications Board.

Prawa autorskie © 2021-2022 autorzy wersji 3.1.0 sylabusa, erraty 3.1.1. i erraty 3.1.2.: Wim Decoutere, István Forgács, Matthias Hamburg, Adam Roman, Jan Sabak, Marc-Florian Wendland.

Prawa autorskie © 2019 autorzy wersji 2019 sylabusa: Graham Bath, Judy McKay, Jan Sabak, Erik van Veenendaal.

Prawa autorskie © 2012 autorzy wersji 2012 sylabusa: Judy McKay, Mike Smith, Erik van Veenendaal.

Wszelkie prawa zastrzeżone.

Autorzy niniejszym przenoszą autorskie prawa majątkowe na ISTQB®. Autorzy (jako obecni posiadacze autorskich praw majątkowych) oraz ISTQB® (jako przyszły posiadacz autorskich praw majątkowych) uzgodnili następujące warunki korzystania z dokumentu:

Wyciągi (fragmenty) z niniejszego dokumentu mogą być kopiowane do użytku niekomercyjnego, jeśli zostanie podane źródło. Każdy akredytowany dostawca szkoleń może wykorzystywać ten sylabus jako podstawę dla szkolenia, o ile zachowane są informacje o autorach i ISTQB® jako źródle i właścicielach praw autorskich do sylabusa. Powoływanie się na niniejszy sylabus we wszelkich materiałach reklamowych i promocyjnych dozwolone jest dopiero po uzyskaniu oficjalnej akredytacji materiałów szkoleniowych przyznanej przez uznaną przez ISTQB® Radę Krajową. (w przypadku Polski: od Stowarzyszenia Jakości Systemów Informatycznych) oficjalnej akredytacji materiałów szkoleniowych.

Każda osoba lub grupa osób może używać tego sylabusa jako podstawy dla artykułów i książek, jeśli autorzy i ISTQB® są wskazani jako źródło i właściciele praw autorskich do sylabusa. Każde inne użycie sylabusa jest zabronione bez wcześniejszego uzyskania zgody ISTQB®.

Każda Rada Krajowa uznawana przez ISTQB® może przetłumaczyć ten sylabus pod warunkiem, że powieli i opublikuje wyżej wymienioną informację o prawach autorskich w przetłumaczonej wersji sylabusa.

Tłumaczenie z języka angielskiego wersji beta – BTInfo Biuro Tłumaczeń Informatycznych Przyłuccy Sp. j.
Przeglądy i uaktualnienie do wersji 1.1 przygotował zespół SJSI w składzie: Joanna Kazun, Jan Sabak, Karolina Sekuła, Lucjan Stapp (kierownik zespołu), Adam Ścierski.
Przeglądy i uaktualnienie do wersji 3.1.0 przygotował zespół SJSI w składzie: Monika Petri-Starego, Adam Roman, Lucjan Stapp (kierownik zespołu).

Historia zmian

Wersja	Data	Uwagi
Wersja 2012	19.10.2012 r.	Opublikowanie przez ISTQB®
Wersja 2019 1.0.	18.10.2019 r.	Opublikowanie przez ISTQB®
Wersja 2019 1.1.	19.12.2019 r.	Opublikowanie przez ISTQB®
Wersja 3.1.0.	03.03.2021 r.	Drobne poprawki; przepisanie rozdziału 3.2.3., poprawienie błędów w tekście
Wersja 3.1.1	15.05.2021 r.	Informacja o prawach autorskich została dostosowana do aktualnych standardów ISTQB®.
Wersja 3.1.2.	31.01.2022 r.	Errata: poprawiono drobne błędy formatowania, gramatyki i słownictwa

Szczegółowy opis zmian w stosunku do poprzedniej wersji sylabusa można znaleźć w nocie wydania (Release Notes).

Historia zmian wersji polskiej

Wersja	Data	Uwagi
Wersja 0.1.	01.03.2020 –31.03.2020 r.	Tłumaczenie wersji beta BTInfo Biuro Tłumaczeń Informatycznych Przyłuccy sp. j.
Wersja 1.0.	01.04.2020 – 30.05.2020 r.	Przeglądy tłumaczenia – Zespół SJSI
Wersja 2.0.	Marzec 2021 r.	Dostosowanie polskiego tłumaczenia do angielskiej wersji 3.1.0 sylabusa
Wersja 3.1.0	12.04.2021 r.	Opublikowanie sylabusa
Wersja 3.1.0.1.	07.03.2023 r.	Zmiana zasad numeracji wersji sylabusa (ostatnia cyfra oznacza zmiany w polskiej wersji językowej). Zmiana tłumaczenia terminu zgodność – poprawiono na „kompatybilność”: R. 4 – słowa kluczowe, 4.1. zmiana tego terminu w tabeli charakterystyk funkcjonalnych, p. 8 zał. A, Indeks.
Wersja 3.1.2.2.	15.05.2023 r.	Poprawa tłumaczenia terminu „bezpieczeństwo” na „zabezpieczenia” w: 3.3.3., 4.1. (lista charakterystyk i podcharakterystyk jakościowych), 6.3.2., Indeks. Dostosowanie informacji o prawach autorskich do aktualnych standardów ISTQB (zgodnie z wersją 3.1.1. oryginału). Poprawienie drobnych błędów dotyczących słownictwa (zgodnie z wersją 3.1.2. oryginału). Pozostałe zmiany: - p. 1.4. drugi akapit z punktorami: było „przedmiotami testów”, poprawiono na „elementami testów”, - p. 1.4.2. zmiana treści ósmego akapitu (nt. trudno dostępnej infrastruktury testowej), - p. 2.1. oraz p. 2.3. szósta kropka: ostanie zdanie z „bezpieczeństwie funkcjonalnym” poprawiono na „bezpieczeństwie”, - p. 3.2.2. dodano „lub technicznego” punktu widzenia (w „Obszar zastosowania”, trzecie zdanie), - p. 3.2.3. zmiana treści 6-go akapitu, zmiana drugiego zdania w części „Pokrycie”

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

		- 3.2.4. w części „Pokrycie” czwarty akapit wykreślono „i pokrywające zbiory testów”, - p. 3.2.7. „testów opartych na scenariuszach” zastąpiono „testów opartych na doświadczeniu”, - p. 3.2.8. zmiana treści ostatniego zdania, - p. 3.4. „pokryciu” zmieniono na „osiągnięciu celów testów”. - p. 4.2.5. ankiety i kwestionariusze dotyczące użyteczności zmieniono na „… dotyczące użytkowników”, - p. 4.2.5.1. dodanie zapisów dotyczących aspektów użyteczności - p. 5.2.1. „główne zachowanie” zmieniono na „podstawowe zachowanie”, - p. 6.1. określenie „automatyzacja oparta na słowach kluczowych” zastąpiono określeniem „testowanie oparte….”. - p. 6.2. zmiana tytułu podrozdziału z „automatyzacja oparta na słowach kluczowych” na „testowanie oparte na słowach kluczowych” oraz zmiana z „przypadków testowych” na „skryptów testowych” - p. 7.3. w wykazie książek dodano [Mosley93].
Wersja 3.1.2.3	27.07.2023 r.	Zmiana tłumaczenia podcharakterystyki jakościowej z „możliwość ponownego wykorzystania” na „łatwość ponownego użycia” (w: 4.1.)
Wersja 3.1.2.4.	10.10.2023 r.	Zmiana tłumaczenia podcharakterystyk jakościowych: podlegający analizie – analizowalność, łatwość ponownego użycia - możliwość ponownego wykorzystania, estetyka interfejsu użytkownika (atrakcyjność) – estetyka interfejsu użytkownika (w: 4.1.)

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

Spis treści

Historia zmian ... 3
Historia zmian wersji polskiej ... 3
Spis treści ... 5
Podziękowania ... 8
0. Wstęp ... 10
0.1. Cel niniejszego sylabusa ... 10
0.2. Certyfikowany tester — poziom zaawansowany w testowaniu oprogramowania ... 10
0.3. Cele nauczania objęte egzaminem i poziomy poznawcze ... 10
0.4. Egzamin na poziomie zaawansowanym dla analityka testów ... 11
0.5. Wymagania stawiane kandydatom przystępującym do egzaminu ... 11
0.6. Oczekiwane doświadczenie ... 11
0.7. Akredytacja szkoleń ... 11
0.8. Poziom szczegółowości informacji ... 11
0.9. Struktura sylabusa ... 12
1. Zadania analityka testów w procesie testowym — 150 minut ... 13
1.1. Wprowadzenie ... 14
1.2. Testowanie w cyklu wytwarzania oprogramowania ... 14
1.3. Analiza testów ... 16
1.4. Projektowanie testów ... 17
1.4.1. Przypadki testowe niskiego i wysokiego poziomu ... 17
1.4.2. Projektowanie przypadków testowych ... 19
1.5. Implementacja testów ... 20
1.6. Wykonywanie testów ... 22
2. Zadania analityka testów w testowaniu opartym na ryzyku — 60 minut ... 23
2.1. Wprowadzenie ... 24
2.2. Identyfikacja ryzyka ... 24
2.3. Ocena ryzyka ... 25
2.4. Łagodzenie ryzyka ... 25
2.4.1. Ustalanie priorytetów testów ... 26
2.4.2. Dostosowywanie testów na potrzeby przyszłych cykli testowania ... 26
3. Techniki testowania — 630 minut ... 27
3.1. Wprowadzenie ... 28
3.2. Czarnoskrzynkowe techniki testowania ... 28
3.2.1. Podział na klasy równoważności ... 28
3.2.2. Analiza wartości brzegowych ... 30
3.2.3. Testowanie w oparciu o tablicę decyzyjną ... 31

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

3.2.4. Testowanie przejść pomiędzy stanami ... 33
3.2.5. Technika drzewa klasyfikacji ... 35
3.2.6. Testowanie sposobem par ... 36
3.2.7. Testowanie oparte na przypadkach użycia ... 37
3.2.8. Łączenie technik ... 38
3.3. Techniki testowania oparte na doświadczeniu ... 38
3.3.1 Zgadywanie błędów ... 39
3.3.2. Testowanie w oparciu o listę kontrolną ... 40
3.3.3. Testowanie eksploracyjne ... 41
3.3.4. Techniki testowania oparte na defektach ... 42
3.4. Zastosowanie najbardziej odpowiedniej techniki ... 43
4. Testowanie charakterystyk jakościowych oprogramowania — 180 minut ... 44
4.1. Wprowadzenie ... 45
4.2. Charakterystyki jakościowe w testowaniu w dziedzinie biznesowej ... 46
4.2.1. Testowanie poprawności funkcjonalnej ... 46
4.2.2. Testowanie adekwatności funkcjonalnej ... 46
4.2.3. Testowanie kompletności funkcjonalnej ... 47
4.2.4. Testowanie współdziałania ... 47
4.2.5. Ocena użyteczności ... 48
4.2.6. Testowanie przenaszalności ... 50
5. Przeglądy — 120 minut ... 52
5.1. Wprowadzenie ... 53
5.2. Korzystanie z list kontrolnych podczas przeglądów ... 53
5.2.1. Przeglądy wymagań ... 53
5.2.2. Przeglądy historyjek użytkownika ... 54
5.2.3. Dostosowywanie list kontrolnych ... 55
6. Narzędzia testowe i automatyzacja testów — 90 minut ... 56
6.1. Wprowadzenie ... 57
6.2. Testowanie oparte na słowach kluczowych ... 57
6.3. Rodzaje narzędzi testowych ... 58
6.3.1. Narzędzia do projektowania testów ... 58
6.3.2. Narzędzia do przygotowywania danych testowych ... 58
6.3.3 Narzędzia do wykonywania testów automatycznych ... 59
7. Dokumenty pomocnicze ... 60
7.1. Standardy ... 60
7.2. Dokumenty ISTQB® i IREB® ... 60
7.3. Książki i artykuły ... 60

© International Software Testing Qualifications Board
Stowarzyszenie Jakości Systemów Informatycznych

7.4. Inne dokumenty pomocnicze 61
8. Załącznik A 62
9. Indeks 63

© International Software Testing Qualifications Board
Stowarzyszenie Jakości Systemów Informatycznych

Podziękowania

Niniejszy dokument został opracowany przez Grupę Roboczą ds. Poziomu Zaawansowanego i Eksperckiego (Advanced and Expert Level Working Group) działającą w ramach ISTQB® w następującym składzie: Mette Bruhn-Pedersen (kierownik zespołu), Matthias Hamburg (Product Owner), Wim Decoutere, István Forgács, Adam Roman, Jan Sabak, Marc-Florian Wendland (autorzy).

Zespół składa podziękowania Paulowi Weymouth i Richardowi Green za redakcję techniczną tekstu, Garemu Mogyorodi za weryfikację zgodności ze Słownikiem terminów testowych ISTQB®, Radom Krajowym za sugestie i wskazówki.

W procesie weryfikacji, zgłaszania uwag i głosowania nad niniejszym sylabusem uczestniczyły następujące osoby: Gery Ágnecz, Armin Born, Chenyifan, Klaudia Dussa-Zieger, Chen Geng (Kevin), Istvan Gercsák, Richard Green, Ole Chr. Hansen, Zsolt Hargitai, Andreas Hetz, Tobias Horn, Joan Killeen, Attila Kovacs, Rik Marselis, Marton Matyas, Blair Mo, Gary Mogyorodi, Ingvar Nordström, Tal Pe'er, Palma Polyak, Nishan Portoyan, Meile Posthuma, Stuart Reid, Murian Song, Péter Sótér, Lucjan Stapp, Benjamin Timmermans, Chris van Bael, Stephanie van Dijck, Paul Weymouth.

Niniejszy dokument został opublikowany przez ISTQB® w dniu 23 lutego 2021 r.

Następnie Tal Pe'er, Stuart Reid, Marc-Florian Wendland i Matthias Hamburg zasugerowali formalne i gramatyczne udoskonalenia oraz poprawki w zakresie słownictwa, które zostały wdrożone i opublikowane w Erracie 3.1.1 i 3.1.2.

Wersja 2019 niniejszego sylabusa została opracowana przez Grupę Roboczą ds. Poziomu Zaawansowanego w składzie: Graham Bath, Judy McKay, Mike Smith.

W procesie weryfikacji, zgłaszania uwag i głosowania nad sylabusem w wersji 2019 wzięły udział następujące osoby: Laura Albert, Markus Beck, Henriett Braunné Bokor, Francisca Cano Ortiz, Guo Chaonian, Wim Decoutere, Milena Donato, Klaudia Dussa-Zieger, Melinda Eckrich-Brajer, Péter Földházi Jr, David Frei, Chen Geng, Matthias Hamburg, Zsolt Hargitai, Zhai Hongbao, Tobias Horn, Ágota Horváth, Beata Karpinska, Attila Kovács, József Kreisz, Dietrich Leimsner, Ren Liang, Claire Lohr, Ramit Manohar Kaul, Rik Marselis, Marton Matyas, Don Mills, Blair Mo, Gary Mogyorodi, Ingvar Nordström, Tal Peer, Pálma Polyák, Meile Posthuma, Lloyd Roden, Adam Roman, Abhishek Sharma, Péter Sótér, Lucjan Stapp, Andrea Szabó, Jan te Kock, Benjamin Timmermans, Chris Van Bael, Erik van Veenendaal, Jan Versmissen, Carsten Weise, Robert Werkhoven, Paul Weymouth.

Wersja 2012 tego sylabusa została opracowana przez podgrupę “zaawansowany analityk testów” Grupy Roboczej ds. Poziomu Zaawansowanego w składzie: Judy McKay (przewodnicząca), Mike Smith, Erik van Veenendaal.

W czasie, gdy sylabus w wersji 2012 został ukończony, skład Grupy Roboczej ds. Poziomu Zaawansowanego był następujący (w kolejności alfabetycznej): Graham Bath, Rex Black, Maria Clara Choucair, Debra Friedenberg, Bernard Homès (z-ca przewodniczącego), Paul Jorgensen, Judy McKay, Jamie Mitchell, Thomas Mueller, Klaus Olsen, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Jan Sabak, Hans Schaefer, Mike Smith (przewodniczący), Geoff Thompson, Erik van Veenendaal, Tsuyoshi Yumoto.

W procesie weryfikacji, zgłaszania uwag i głosowania nad sylabusem w wersji 2012 uczestniczyły następujące osoby: Graham Bath, Arne Becher, Rex Black, Piet de Roo, Frans Dijkman, Mats Grindal, Kobi Halperin, Bernard Homès, Maria Jönsson, Junfei Ma, Eli Margolin, Rik Marselis, Don Mills, Gary Mogyorodi, Stefan Mohacsi, Reto Mueller, Thomas Mueller, Ingvar Nordstrom, Tal Pe'er, Raluca Madalina Popescu, Stuart Reid, Jan Sabak, Hans Schaefer, Marco Sogliani, Yaron Tsubery, Hans Weiberg, Paul Weymouth,

© International Software Testing Qualifications Board
Stowarzyszenie Jakości Systemów Informatycznych

Chris van Bael, Jurian van der Laar, Stephanie van Dijk, Erik van Veenendaal, Wenqiang Zheng, Debi Zylbermann.

© International Software Testing Qualifications Board
Stowarzyszenie Jakości Systemów Informatycznych

0. Wstęp

0.1. Cel niniejszego sylabusa

Niniejszy sylabus stanowi podstawę egzaminu International Software Testing Qualification Board dla analityków testów na poziomie zaawansowanym. ISTQB® udostępnia sylabus:

1. Radom Krajowym (National Boards) w celu tłumaczenia na języki lokalne i dokonania akredytacji dostawców szkoleń (Rady Krajowe mogą dostosowywać sylabus do potrzeb danego języka i dodawać odwołania do literatury w celu uwzględnienia publikacji lokalnych),
2. komisjom egzaminacyjnym (Exam Boards) jako podstawę do formułowania pytań egzaminacyjnych w językach lokalnych, odpowiadających celom nauczania danego 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 (w ramach szkoleń zorganizowanych lub przygotowania indywidualnego),
5. międzynarodowej społeczności specjalistów w dziedzinie inżynierii oprogramowania i systemów w celu rozwijania zawodu testera oprogramowania i systemów oraz opracowywania książek i artykułów.

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

0.2. Certyfikowany tester — poziom zaawansowany w testowaniu oprogramowania

Kwalifikacja na poziomie zaawansowanym obejmuje w ramach ścieżki głównej trzy odrębne sylabusy związane z następującymi rolami:

- Kierownik testów
- Analityk testów
- Techniczny analityk testów.

„Wprowadzenie do poziomu zaawansowanego ISTQB®" to oddzielny dokument [ISTQB_AL_OVIEW], w którym zawarto następujące informacje:

- cele biznesowe dla każdego sylabusa,
- macierz powiązań między celami biznesowymi a celami nauczania,
- podsumowanie każdego sylabusa,
- powiązania między sylabusami.

0.3. Cele nauczania objęte egzaminem i poziomy poznawcze

Cele nauczania wspierają osiąganie celów biznesowych i stanowią wytyczne do formułowania pytań dla egzaminów certyfikacyjnych „Analityk testów — poziom zaawansowany".

Poziomy wiedzy związane z poszczególnymi celami nauczania przedstawiono na początku każdego rozdziału. Poziomy te sklasyfikowano następująco:

- K2: zrozumieć
- K3: zastosować
- K4: przeanalizować.

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

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

0.4. Egzamin na poziomie zaawansowanym dla analityka testów

Zakres egzaminu umożliwiającego uzyskanie certyfikatu analityka testów na poziomie zaawansowanym opiera się 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 innych sylabusów ISTQB®, norm/standardów i książek, ale ich treść nie może być przedmiotem egzaminu w zakresie wykraczającym poza informacje streszczone w samym sylabusie.

Egzamin ma formę testu wielokrotnego wyboru i składa się z 40 pytań. Do zdania egzaminu niezbędne jest uzyskanie co najmniej 65% punktów możliwych do uzyskania.

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

0.5. Wymagania stawiane kandydatom przystępującym do egzaminu

Przed przystąpieniem do egzaminu certyfikacyjnego dla analityka testów na poziomie zaawansowanym należy zdać egzamin certyfikacyjny związany z sylabusem „Certyfikowany tester — poziom podstawowy”.

0.6. Oczekiwane doświadczenie

Żaden z celów nauczania określonych dla analityka testów nie zakłada posiadania konkretnego doświadczenia.

0.7. Akredytacja szkoleń

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. Akredytowane szkolenie jest uznawane za zgodne z niniejszym sylabusem i może obejmować egzamin ISTQB®.

0.8. Poziom szczegółowości informacji

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

- ogólne cele dydaktyczne opisujące założenia poziomu zaawansowanego w odniesieniu do analityków testów,
- wykaz pojęć, które muszą zapamiętać uczestnicy szkolenia,
- cele nauczania w poszczególnych obszarach wiedzy, opisujące efekty kształcenia o charakterze poznawczym,
- opis najważniejszych pojęć, w tym odwołania do źródeł (takich jak uznane publikacje oraz normy lub 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 zaawansowanym. W sylabusie skupiono się na zagadnieniach, które mogą mieć zastosowanie we wszystkich projektach wytwarzania oprogramowania, w tym w projektach zwinnego wytwarzania oprogramowania. Sylabus nie zawiera żadnych konkretnych celów nauczania związanych z określonym cyklem wytwarzania oprogramowania, natomiast omówiono w nim, w jaki sposób wprowadzone pojęcia

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

można zastosować do modelu zwinnego wytwarzania oprogramowania, do innych modeli iteracyjnych i przyrostowych oraz do modeli sekwencyjnych.

0.9. Struktura sylabusa

Sylabus zawiera sześć rozdziałów, których treść może być przedmiotem egzaminu. Nagłówek najwyższego poziomu zawiera informację o minimalnym czasie trwania szkolenia dla każdego rozdziału (wykładów i ćwiczeń) – nie podano czasu trwania podrozdziałów i mniejszych jednostek redakcyjnych. W przypadku akredytowanych szkoleń na przekazanie wiedzy zawartej w sylabusie potrzeba co najmniej 20 godzin i 30 minut wykładów. Czas ten podzielono na poszczególne rozdziały w następujący sposób:

- Rozdział 1: Zadania analityka testów w procesie testowym (150 minut)
- Rozdział 2: Zadania analityka testów w testowaniu opartym na ryzyku (60 minut)
- Rozdział 3: Techniki testowania (630 minut)
- Rozdział 4: Testowanie charakterystyk jakościowych oprogramowania (180 minut)
- Rozdział 5: Przeglądy (120 minut)
- Rozdział 6: Narzędzia testowe i automatyzacja testów (90 minut).

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

1. Zadania analityka testów w procesie testowym — 150 minut

Słowa kluczowe
analiza testów, dane testowe, harmonogram testów, implementacja testów, kryterium wyjścia, podstawa testów, procedura testowa, projekt testów, przypadek testowy niskiego poziomu, przypadek testowy wysokiego poziomu, test, warunek testowy, wykonywanie testu, zestaw testowy

Cele nauczania związane z zadaniami analityka testów w procesie testowym

1.1. Wstęp
Nie określono celów nauczania.

1.2. Testowanie w cyklu wytwarzania oprogramowania
TA-1.2.1. (K2) Kandydat potrafi wyjaśnić, w jaki sposób i z jakich przyczyn czas i zakres zaangażowania analityka testów różnią się w różnych modelach cyklu wytwarzania oprogramowania.

1.3. Analiza testów
TA-1.3.1. (K2) Kandydat potrafi omówić zadania, które powinien wykonać analityk testów w trakcie analizy testów.

1.4. Projektowanie testów
TA-1.4.1. (K2) Kandydat potrafi wyjaśnić, dlaczego interesariusze powinni rozumieć warunki testowe.
TA-1.4.2. (K4) Kandydat potrafi wybrać właściwy poziom projektowania przypadków testowych w danym scenariuszu projektowym (przypadki testowe wysokiego lub niskiego poziomu).
TA-1.4.3. (K2) Kandydat potrafi omówić zagadnienia, które należy uwzględnić podczas projektowania przypadków testowych.

1.5. Implementacja testów
TA-1.5.1. (K2) Kandydat potrafi omówić zadania, które powinien wykonać analityk testów w trakcie implementacji testów.

1.6. Wykonywanie testów
TA-1.6.1. (K2) Kandydat potrafi omówić zadania, które powinien wykonać analityk testów w trakcie wykonywania testów.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

1.1. Wprowadzenie

W sylabusie ISTQB® poziomu podstawowego opisano następujące czynności procesu testowego:

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

W niniejszym sylabusie dokładniej opisano czynności, które są szczególnie istotne z punktu widzenia analityka testów. Pozwala to zwiększyć szczegółowość procesu testowego i lepiej dostosować go do potrzeb różnych modeli cyklu wytwarzania oprogramowania.

Określenie właściwych testów oraz ich zaprojektowanie, implementacja, a następnie wykonanie, to najważniejsze obszary działania analityka testów. Zrozumienie pozostałych kroków procesu testowego jest istotne, ale na ogół praca analityka testów obejmuje przede wszystkim następujące czynności:

- analiza testów,
- projektowanie testów,
- implementacja testów,
- wykonywanie testów.

Inne czynności wchodzące w skład procesu testowego zostały odpowiednio przedstawione w sylabusie poziomu podstawowego i nie ma potrzeby ich dalszego opisywania na poziomie zaawansowanym.

1.2. Testowanie w cyklu wytwarzania oprogramowania

W ramach definiowania strategii testów należy wziąć pod uwagę stosowany cykl wytwarzania oprogramowania. Między modelami cyklu wytwarzania oprogramowania istnieją różnice dotyczące momentu zaangażowania analityka testów, stopnia tego zaangażowania, wymaganego od analityka testów nakładu czasu, dostępnych dla niego informacji i wreszcie oczekiwań względem osoby występującej w tej roli. Analityk testów musi wiedzieć, jakie informacje powinien przekazywać osobom pełniącym inne role w organizacji. Dotyczy to takich obszarów jak:

- inżynieria wymagań i zarządzanie wymaganiami — informacje zwrotne na temat przeglądów wymagań,
- zarządzanie projektem — informacje wejściowe do harmonogramów,
- zarządzanie konfiguracją i zarządzanie zmianami — wyniki weryfikacji wersji poprzez testowanie, informacje na temat kontroli wersji,
- wytwarzanie oprogramowania — powiadomienia o znalezionych defektach,
- pielęgnacja oprogramowania — zgłoszenia defektów, efektywność usuwania defektów, testowanie potwierdzające,
- wsparcie techniczne — dokładne dokumentowanie sposobów ominięcia znanych problemów,
- tworzenie dokumentacji technicznej (np. specyfikacji projektu bazy danych, dokumentacji środowiska testowego) — informacje wejściowe do tych dokumentów oraz przeglądy techniczne dokumentów.

Czynności testowe muszą być dostosowane do wybranego modelu cyklu wytwarzania oprogramowania (SDLC – *Software Development Life Cycle*), który może mieć charakter sekwencyjny, iteracyjny, przyrostowy lub stanowić hybrydę tychże. Na przykład w sekwencyjnym modelu V proces testowy

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

zastosowany na poziomie testów systemowych można dopasować do procesu wytwarzania oprogramowania w następujący sposób:

- Planowanie testów systemowych odbywa się równolegle z planowaniem projektu, a monitorowanie testów i nadzór nad testami jest prowadzone do czasu ich ukończenia. Ma to wpływ na informacje dotyczące harmonogramu działań przekazywane przez analityka testów osobom odpowiedzialnym za zarządzanie projektem.
- Analiza i projektowanie testów systemowych odbywają się równolegle z tworzeniem takich dokumentów jak specyfikacja wymagań systemowych, specyfikacja projektu systemu i architektury (specyfikacja wysokiego poziomu) oraz specyfikacja projektu modułów (specyfikacja niskiego poziomu).
- Implementacja środowiska testowego dla testów systemowych może się rozpocząć podczas projektowania systemu, ale główna część prac odbywa się równolegle z kodowaniem i testami modułowymi, a czynności związane z implementacją testów systemowych często kończą się zaledwie kilka dni przed rozpoczęciem wykonywania testów systemowych.
- Wykonywanie testów systemowych rozpoczyna się po spełnieniu lub, jeśli to konieczne, podjęciu decyzji o uchyleniu kryteriów wejścia, co zwykle oznacza, że co najmniej testy modułowe, a często również testy integracyjne, mają spełnione kryteria wyjścia. Wykonywanie testów systemowych jest kontynuowane do chwili spełnienia kryteriów wyjścia testów systemowych.
- Czynności związane z ukończeniem testów systemowych wykonywane są po spełnieniu ich kryteriów wyjścia.

W modelach iteracyjnych i przyrostowych kolejność wykonywania zadań może być inna, a niektóre z nich mogą się w ogóle nie pojawiać. Na przykład w modelu iteracyjnym w poszczególnych iteracjach może być stosowany jedynie pewien ograniczony zestaw czynności testowych. Analiza testów, projektowanie, implementacja i wykonanie mogą być prowadzone w każdej iteracji, natomiast planowanie na wysokim poziomie odbywa się na początku projektu, a realizacja zadań związanych z ukończeniem testów — na końcu.

W zwinnym wytwarzaniu oprogramowania zazwyczaj funkcjonują mniej sformalizowane procesy, a kontakty robocze między interesariuszami projektu są znacznie bliższe, co ułatwia wprowadzanie zmian w projekcie. W takich projektach może nie występować szczegółowo zdefiniowana rola analityka testów. Dokumentacja testowa jest mniej obszerna, a wymiana informacji następuje częściej, ma za to węższy zakres.

W zwinnym wytwarzaniu oprogramowania testowanie uwzględnia się już na wstępnych etapach. Rozpoczyna się ono w fazie wytwarzania produktu, kiedy programiści wykonują wstępne działania związane z tworzeniem architektury i projektowaniem. Przeglądy raczej nie są sformalizowane, ale odbywają się w trybie ciągłym wraz z ewolucją oprogramowania. Zaangażowania w prace testowe oczekuje się przez cały czas trwania projektu, a zadania analityka testów powinni realizować członkowie zespołu.

Modele iteracyjne/przyrostowe pokrywają szeroki zakres modeli wytwarzania oprogramowania — od zwinnego wytwarzania oprogramowania, w którym oczekuje się ciągłych zmian wraz z ewolucją wymagań klienta, po modele hybrydowe, np. wytwarzanie iteracyjne/przyrostowe połączone z podejściem modelu V. W takich modelach hybrydowych analityk testów powinien być zaangażowany w aspekty planowania i projektowania czynności sekwencyjnych, a następnie przyjąć bardziej interaktywną rolę podczas dalszych czynności iteracyjnego/przyrostowego wytwarzania.

Niezależnie od stosowanego modelu cyklu wytwarzania oprogramowania analityk testów powinien rozumieć oczekiwania dotyczące czasu i stopnia zaangażowania w projekt. Analityk testów dostarcza najbardziej efektywny wkład w jakość oprogramowania poprzez dostosowanie swoich działań i wybór momentu zaangażowania w określony model cyklu wytwarzania oprogramowania, zamiast dopasowanie się do predefiniowanej w modelu roli.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

1.3. Analiza testów

W fazie planowania testów zostaje zdefiniowany zakres projektu testowego. Na etapie analizy testów, na jego podstawie analityk testów:

- analizuje podstawę testów,
- identyfikuje różne typy defektów w podstawie testów,
- identyfikuje i priorytetyzuje warunki testowe i funkcje, które mają być testowane,
- rejestruje dwukierunkowe powiązania między poszczególnymi elementami podstawy testów a powiązanymi warunkami testowymi,
- wykonuje zadania związane z testowaniem opartym na ryzyku (patrz Rozdział 2.).

Aby analityk testów mógł efektywnie przeprowadzić analizę testów, powinny być spełnione następujące kryteria wejścia:

- Istnieje zapis wiedzy (np. wymagania, historyjki użytkownika) na temat przedmiotu testów, który może stanowić podstawę testów (punkt 1.4.2. i podrozdział 2.2. sylabusa [ISTQB_FL_SYL] zawiera listę innych możliwych źródeł podstawy testów).
- Taka podstawa testów przeszła przegląd z wystarczająco dobrym wynikiem i została odpowiednio zmodyfikowana po przeglądzie. Należy zauważyć, że jeśli mają zostać zdefiniowane przypadki testowe wysokiego poziomu (patrz punkt 1.4.1.), podstawa testów może nie być jeszcze w pełni zdefiniowana. W zwinnym wytwarzaniu oprogramowania cykl przeglądu będzie mieć charakter iteracyjny, ponieważ historyjki użytkownika są uszczegóławiane na początku każdej iteracji.
- Zatwierdzone w budżecie środki oraz przyjęty harmonogram pozwalają wykonać pozostałe prace związane z testowaniem danego przedmiotu testów.

Warunki testowe identyfikuje się z reguły poprzez analizę podstawy testów i celów testowania (zgodnie z definicją przyjętą w procesie planowania testów). W pewnych sytuacjach, gdy dokumentacja jest nieaktualna lub nie istnieje, warunki testowe można ustalić po dyskusji z odpowiednimi interesariuszami (np. w trakcie warsztatów lub podczas planowania iteracji). W zwinnym wytwarzaniu oprogramowania kryteria akceptacji definiowane w ramach historyjek użytkownika są często wykorzystywane jako podstawa projektowania testów.

Warunki testowe są zwykle specyficzne dla danego obiektu testowego, analityk testów powinien jednak uwzględnić pewne standardowe uwarunkowania:

- Zazwyczaj warto definiować warunki testowe na różnych poziomach szczegółowości. Na początku identyfikuje się warunki wysokiego poziomu w celu określenia ogólnych obszarów testowania, np. „funkcjonalność ekranu X”. Następnie identyfikuje się bardziej szczegółowe warunki, stanowiące podstawę przypadków testowych, takie jak „ekran X odrzuca numer konta, który jest o jedną cyfrę krótszy niż poprawny numer”. Takie hierarchiczne podejście do definiowania warunków testowych ułatwia zapewnienie odpowiedniego pokrycia obiektów wysokiego poziomu. Umożliwia ono również analitykowi testów rozpoczęcie pracy nad warunkami testowymi wysokiego poziomu związanymi z historyjkami użytkownika, które jeszcze nie zostały uszczegółowione.
- Jeżeli zdefiniowano ryzyka produktowe, należy zidentyfikować warunki testowe dotyczące każdego z czynników ryzyka i powiązać je z odpowiednimi elementami ryzyka.

Zastosowanie technik testowania (określonych w strategii testów i/lub planie testów) może ułatwić przeprowadzenie czynności analizy testów i może wspierać realizację następujących celów:

- identyfikacja warunków testowych,
- zmniejszenie prawdopodobieństwa pominięcia ważnych warunków testowych,
- zdefiniowanie dokładniejszych i bardziej poprawnych warunków testowych.

Po zdefiniowaniu i uszczegółowieniu warunków testowych należy przeprowadzić ich przegląd z udziałem interesariuszy, aby zyskać pewność, że wymagania są jednoznacznie zinterpretowane, a proces testowania odpowiada celom projektu.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

Po zakończeniu analizy testów dla danego obszaru (np. określonej funkcji) analityk testów powinien wiedzieć, jakie konkretne testy należy zaprojektować w tym obszarze.

1.4. Projektowanie testów

W kolejnym kroku procesu testowego analityk testów projektuje testy, które mają zostać zaimplementowane i wykonane w ramach zakresu testowania ustalonego podczas planowania testów. Czynności wykonywane w ramach projektowania testów:

- Określenie, dla których obszarów testowych odpowiednie byłyby przypadki testowe niskiego poziomu, a dla których przypadki testowe wysokiego poziomu.
- Określenie technik testowania, które pozwolą uzyskać wymagane pokrycie. Techniki, z których można skorzystać, ustala się w trakcie planowania testów.
- Wykorzystanie technik testowania do zaprojektowania przypadków testowych i zestawów testowych pokrywających zidentyfikowane warunki testowe.
- Zidentyfikowanie danych testowych niezbędnych do obsługi warunków testowych i przypadków testowych.
- Zaprojektowanie środowiska testowego oraz zidentyfikowanie wszelkich niezbędnych elementów infrastruktury, w tym narzędzi.
- Stworzenie możliwości dwukierunkowego śledzenia powiązań (np. między podstawą testów, warunkami testowymi a przypadkami testowymi).

W całym procesie od analizy i projektowania po implementację i wykonywanie testów należy stosować kryteria wyznaczania priorytetów ustalone podczas analizy ryzyka i planowania testów.

W zależności od typów projektowanych testów jednym z kryteriów wejścia do fazy projektowania testów może być dostępność narzędzi wykorzystywanych do projektowania testów.

Podczas projektowania testów analityk testów musi uwzględnić co najmniej następujące zagadnienia:

- W przypadku niektórych elementów testowych lepiej sprawdza się zdefiniowanie tylko warunków testowych, bez schodzenia do poziomu definiowania skryptów testowych, w których zapisana jest sekwencja instrukcji niezbędnych do wykonania testu. W takich sytuacjach należy zdefiniować warunki testowe jako wytyczne do testowania nie-skryptowego (nie opartego na skryptach testowych).
- Należy jasno określić kryteria zaliczenia i niezaliczenia takiego testu.
- Należy projektować testy tak, aby były zrozumiałe również dla innych testerów, a nie tylko dla autora. Jeżeli to nie autor będzie wykonywać dany test, inni testerzy będą musieli odczytać i zrozumieć zdefiniowane testy, aby zrozumieć cele testowania i względną ważność testu.
- Testy muszą być również zrozumiałe dla innych interesariuszy, np. programistów (którzy mogą dokonywać przeglądów testów) oraz dla audytorów (których akceptacja może być wymagana).
- Testy powinny pokrywać wszelkie interakcje z przedmiotem testów, a nie tylko interakcje użytkowników za pośrednictwem widocznego interfejsu. Mogą to być na przykład interakcje z innymi systemami i zdarzenia techniczne lub fizyczne. Więcej informacji na ten temat można znaleźć w dokumencie [IREB_CPRE].
- Testy należy projektować tak, aby można było przetestować interfejsy między poszczególnymi elementami testowymi, a także zachowanie poszczególnych elementów testów.
- Podczas projektowania testów należy pamiętać o priorytetach i zrównoważeniu zadań, z uwzględnieniem poziomów ryzyka i wartości biznesowej.

1.4.1. Przypadki testowe niskiego i wysokiego poziomu

Jednym z zadań analityka testów jest ustalenie poziomów przypadków testowych najbardziej odpowiednich w danej sytuacji. Przypadki testowe niskiego i wysokiego poziomu zostały opisane w dokumencie [ISTQB_FL_SYL]. Poniżej przedstawiono listy zalet i wad obu rodzajów przypadków testowych.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

Zalety przypadków testowych niskiego poziomu:

- Mniej doświadczeni testerzy mogą skorzystać ze szczegółowych informacji dostępnych w projekcie. Przypadki testowe niskiego poziomu zawierają wszystkie szczegółowe informacje i procedury potrzebne testerowi do wykonania przypadku testowego (w tym wymagania dotyczące danych) i do zweryfikowania rzeczywistych rezultatów.
- Testy mogą być ponownie wykonywane przez różne osoby, a każdy z testerów powinien uzyskać taki sam wynik testu.
- Możliwe jest wykrycie nieoczywistych defektów w podstawie testów.
- Przyjęty poziom szczegółowości umożliwia niezależną weryfikację testów, np. w trakcie audytu, jeśli to jest wymagane.
- Może to zmniejszyć czas poświęcany na implementację przypadków testowych dla testów automatycznych.

Wady przypadków testowych niskiego poziomu:

- Zarówno tworzenie, jak i pielęgnacja przypadków testowych tego typu, może wiązać się ze znaczną pracochłonnością.
- Przypadki testowe niskiego poziomu zwykle ograniczają swobodę działań testera podczas ich wykonywania.
- Wymagają tego, by podstawa testów została dobrze zdefiniowana.
- Śledzenie powiązań przypadków testowych niskiego poziomu z warunkami testowymi może być bardziej pracochłonne niż śledzenie powiązań przypadków wysokiego poziomu.

Zalety przypadków testowych wysokiego poziomu:

- Przypadki testowe wysokiego poziomu zawierają wskazówki, co ma zostać przetestowane, a także umożliwiają analitykowi testów zastosowanie różnych zestawów danych, a nawet procedur wykonania testu.
- Przypadki testowe wysokiego poziomu mogą zapewnić lepsze pokrycie czynników ryzyka niż przypadki niskiego poziomu, ponieważ przy każdym wykonaniu będą nieco inne.
- Przypadki tego rodzaju można zdefiniować we wczesnych etapach procesu specyfikowania wymagań.
- W trakcie wykonywania testów wykorzystywane jest doświadczenie analityka testów związane zarówno z samym procesem testowania, jak i z przedmiotem testów.
- Przypadki testowe wysokiego poziomu można definiować w sytuacji, gdy nie jest wymagane opracowanie szczegółowej, formalnej dokumentacji.
- Przypadki testowe wysokiego poziomu lepiej nadają się do wykorzystania w różnych cyklach testowych, jeśli można użyć różnych danych testowych.

Wady przypadków testowych wysokiego poziomu:

- Przypadki odznaczają się mniejszą powtarzalnością, co utrudnia weryfikację testów. Wynika to z braku szczegółowego opisu, który występuje w przypadkach testowych niskiego poziomu.
- Do wykonania takich przypadków testowych mogą być potrzebni testerzy z nieco większym doświadczeniem.
- Jeśli automatyzacja testów odbywa się na podstawie przypadków testowych wysokiego poziomu, z powodu małej szczegółowości może nastąpić walidacja niepoprawnych rzeczywistych rezultatów, a ponadto niektóre elementy mogą nie zostać sprawdzone.

Przypadki testowe wysokiego poziomu mogą później posłużyć do tworzenia przypadków niskiego poziomu, gdy wymagania zostaną ustabilizowane i doprecyzowane. W takiej sytuacji tworzenie przypadków testowych odbywa się sekwencyjnie z przejściem od przypadków wysokiego poziomu do niskiego poziomu, a do wykonania testów są używane tylko przypadki testowe niskiego poziomu.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

1.4.2. Projektowanie przypadków testowych

Projektowanie przypadków testowych polega na uszczegóławianiu i doprecyzowywaniu zidentyfikowanych warunków testowych zgodnie z technikami testowania (patrz Rozdział 3.). Przypadki testowe powinny być powtarzalne, możliwe do weryfikacji i do powiązania z podstawą testów (np. z wymaganiami).

W ramach projektowania przypadków testowych należy zidentyfikować następujące elementy:

- cel (tj. możliwy do zaobserwowania i zmierzenia wynik wykonania testu),
- warunki wstępne, takie jak wymagania projektowe lub wymagania dotyczące lokalnego środowiska testowego, plany ich dostarczenia, stan systemu przed wykonaniem testu itp.,
- wymagania dotyczące danych testowych (zarówno danych wejściowych do przypadku testowego, jak i danych, które muszą istnieć w systemie, aby można było wykonać przypadek testowy),
- oczekiwane rezultaty i jawne kryteria zaliczenia/niezaliczenia,
- warunki wyjściowe, takie jak zmodyfikowane dane, na które ma wpływ wykonanie przypadku testowego, stan systemu po wykonaniu testów, wyzwalacze dalszego przetwarzania itp.

Szczególną trudność może sprawiać zwłaszcza zdefiniowanie oczekiwanego rezultatu testu. Wyznaczanie go ręcznie jest często uciążliwe i podatne na błędy. W miarę możliwości należy raczej użyć zewnętrznej automatycznej wyroczni testowej lub stworzyć własną. Określając oczekiwany rezultat, testerzy powinni odnieść się nie tylko do danych wyjściowych wyświetlanych na ekranie, ale również do warunków wyjściowych dotyczących danych i środowiska. Przy jasno zdefiniowanej podstawie testów wyznaczenie prawidłowych rezultatów nie powinno sprawić trudności. Jednak dokumentacja podstawy testów może być nieprecyzyjna, sprzeczna, nie pokrywać kluczowych obszarów lub w ogóle nie być dostępna. W takich sytuacjach analityk testów musi dysponować odpowiednią wiedzą dziedzinową lub mieć dostęp do źródła takiej wiedzy. Jednak nawet wtedy, gdy podstawa testów jest dobrze określona, zdefiniowanie oczekiwanych rezultatów mogą utrudnić skomplikowane interakcje złożonych zdarzeń wejściowych i reakcji systemu — nieodzowna jest więc wyrocznia testowa. W zwinnym wytwarzaniu oprogramowania rolę wyroczni testowej może odgrywać właściciel produktu. Wykonywanie przypadków testowych bez możliwości sprawdzenia poprawności rzeczywistych rezultatów zwykle nie niesie ze sobą korzyści, a często powoduje sporządzanie błędnych raportów z testów lub błędne przekonanie o prawidłowym działaniu systemu.

Powyższe czynności mają zastosowanie na wszystkich poziomach testów, choć podstawa testów jest w każdej sytuacji inna. Podczas analizy i projektowania testów należy pamiętać zarówno o poziomie, na jakim ma być wykonywany dany test, jak i o celu testu. Ułatwia to określenie wymaganego poziomu szczegółowości oraz wszelkich wymaganych narzędzi (np. sterowników i zaślepek na poziomie testów modułowych).

Podczas opracowywania warunków i przypadków testowych powstaje zazwyczaj dokumentacja, która stanowi jeden z produktów pracy związanych z testowaniem. W praktyce zakres dokumentowania produktów pracy związanych z testowaniem bywa bardzo różny. Warunkują to następujące czynniki:

- ryzyko projektowe (co musi / nie musi być udokumentowane),
- wartość dodana dokumentacji w projekcie,
- standardy i uregulowania prawne, których należy przestrzegać,
- zastosowany model cyklu wytwarzania oprogramowania lub podejście (np. w modelu zwinnym dąży się do generowania „niezbędnego minimum” dokumentacji),
- wymaganie możliwości śledzenia powiązań między podstawą testów a rezultatami analizy i projektowania testów.

W zależności od zakresu testowania analiza i projektowanie testów mogą obejmować weryfikację jakościowych charakterystyk oprogramowania. Standard ISO 25010 [ISO25010] stanowi tu przydatny materiał pomocniczy. Przy testowaniu systemów sprzętowo-programowych może być konieczne uwzględnienie dodatkowych charakterystyk.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

Czynności analizy i projektowania testów można udoskonalić poprzez wplecenie w nie przeglądów i analizy statycznej. Tak naprawdę przeprowadzanie analizy i projektowania testów same w sobie często stanowią pewną formę testowania statycznego, ponieważ umożliwiają wykrycie problemów w dokumentach podstawy testów podczas tych czynności. Analiza i projektowanie testów w oparciu o specyfikację wymagań są znakomitym przygotowaniem do spotkania przeglądowego (w celu omówienia wymagań). Lektura wymagań niezbędna do opracowania testów oznacza konieczność zrozumienia wymagania i ustalenia sposobu sprawdzenia, czy to wymaganie jest spełnione. Dzięki temu często można odkryć brakujące wymagania oraz wymagania, które są niejasne, nietestowalne lub dla których nie zostały zdefiniowane kryteria akceptacji. Również produkty pracy związane z testowaniem, takie jak przypadki testowe, analizy ryzyka i plany testów, mogą być poddawane przeglądom.

Jeśli testowanie wymaga infrastruktury, która nie jest łatwo dostępna, analityk testów powinien zdefiniować szczegółowe wymagania dotyczące infrastruktury testowej podczas projektowania testu. Jeśli te wymagania nie zostaną spełnione na czas, wdrożenie testów będzie zagrożone ryzykiem z powodu nieoczekiwanego czasu i wysiłku.

Należy pamiętać, że infrastruktura testu obejmuje więcej niż tylko przedmioty testów i testalia. Do wymagań dotyczących infrastruktury mogą należeć wymagania co do pomieszczeń, wyposażenia, personelu, oprogramowania, narzędzi, urządzeń peryferyjnych i komunikacyjnych, uprawnień dla użytkowników i wszelkich innych elementów niezbędnych do wykonania testów.

Kryteria wyjścia dla analizy i projektowania testów mogą być różne w zależności od parametrów projektu, jednak należy rozważyć ujęcie w nich wszystkich elementów omówionych w tym podrozdziale. Istotne jest, aby kryteria wyjścia były mierzalne i umożliwiały zebranie wszystkich informacji i niezbędnych do wykonania kolejnych kroków w procesie oraz przeprowadzenie wszystkich niezbędnych przygotowań.

1.5. Implementacja testów

Implementacja testów polega na przygotowaniu testaliów niezbędnych do wykonania testów na podstawie wyników analizy i projektowania testów. Czynności wykonywane w ramach implementacji testów to:

- Opracowywanie procedur testowych i potencjalnie tworzenie skryptów testów automatycznych.
- Organizowanie procedur testowych i skryptów testów automatycznych (jeśli jakieś istnieją) w zestawy testowe do wykonania w określonym przebiegu testu.
- Konsultowanie z kierownikiem testów priorytetyzacji przypadków testowych i zestawów testowych do wykonania.
- Tworzenie harmonogramu wykonywania testów z uwzględnieniem przydziału zasobów, tak aby było możliwe rozpoczęcie wykonywania przypadków testowych (patrz dokument [ISTQB_FL_SYL], punkt 5.2.4).
- Finalizacja czynności przygotowania danych testowych i środowisk testowych.
- Aktualizacja powiązań między podstawą testów a testaliami takimi jak: warunki testowe, przypadki testowe, procedury testowe, skrypty testowe i zestawy testowe.

W trakcie implementacji testów analityk testów identyfikuje skuteczną kolejność wykonywania przypadków testowych i tworzy procedury testowe.

W ramach tego kroku należy szczegółowo określić ograniczenia i zależności, które mogłyby wymusić uruchamianie testów w określonym porządku. W procedurach testowych są dokumentowane wszystkie warunki wstępne (np. ładowanie danych testowych z repozytorium danych) i działania, które należy wykonać po wykonaniu testów (np. zresetowanie stanu systemu).

Analityk testów identyfikuje procedury testowe i skrypty testów automatycznych, które mogą być pogrupowane (np. wszystkie odnoszą się do testowania konkretnego, wysokopoziomowego procesu

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

biznesowego) i organizuje je w zestawy testowe. Umożliwia to wspólne testowanie powiązanych przypadków testowych.

Analityk testów porządkuje zestawy testowe w ramach harmonogramu wykonania testów w sposób, który skutkuje skutecznym wykonaniem testów. W przypadku zastosowania strategii testów opartej na ryzyku poziom ryzyka będzie podstawowym czynnikiem w określaniu kolejności wykonywania przypadków testowych. Kolejność ta może również zależeć od innych czynników, takich jak dostępność odpowiedniego personelu, sprzętu, danych i testowanych funkcji.

Kod programu często jest publikowany we fragmentach, w związku z czym należy skoordynować testowanie z kolejnością udostępniania poszczególnych elementów oprogramowania. Szczególnie w przypadku iteracyjnych i przyrostowych modeli wytwarzania analityk testów powinien koordynować swoje prace z działaniami zespołu programistów tak, aby elementy oprogramowania były udostępniane do testów w kolejności umożliwiającej ich przetestowanie.

Poziom szczegółowości warunków testowych i przypadków testowych może wpływać na poziom szczegółowości i związaną z nim złożoność prac prowadzonych w ramach implementacji testów. W pewnych przypadkach mają zastosowanie dodatkowe regulacje prawne i produkty związane z testowaniem powinny wówczas być zgodne z odpowiednimi normami, takimi jak amerykański standard DO-178C (ED 12C w Europie). [RTCA DO-178C/ED-12C].

Jak wspomniano powyżej, na ogół do testowania potrzebne są dane testowe. W pewnych przypadkach zestawy danych mogą osiągać duże rozmiary. Podczas implementacji analityk testów tworzy dane wejściowe i dane środowiskowe, które mają zostać załadowane do baz danych i innych repozytoriów. Dane muszą być dostosowane do potrzeb procesu testowania, tj. umożliwiać wykrywanie defektów. Analityk testów może również tworzyć dane, które będą wykorzystywane w testowaniu sterowanym danymi i testowaniu opartym na słowach kluczowych (patrz podrozdział 6.2.) oraz w testowaniu manualnym.

Implementacja testów obejmuje także tworzenie środowisk(-a) testowych(-ego). W ramach tej czynności należy w pełni skonfigurować środowisko(-a) testowe i zweryfikować jego (ich) poprawność. Niezbędne jest środowisko testowe dopasowane do potrzeb testowania, tj. takie, które umożliwi wykrycie defektów w toku kontrolowanego testowania, będzie działać normalnie w przypadku braku awarii i odpowiednio odzwierciedli — o ile jest to wymagane — środowisko produkcyjne lub środowisko użytkownika końcowego dla potrzeb wyższych poziomów testów. Podczas wykonywania testów mogą się okazać konieczne modyfikacje środowiska testowego spowodowane nieprzewidzianymi zmianami, rezultatami testów lub innymi uwarunkowaniami. Jeżeli takie modyfikacje zostaną wprowadzone podczas wykonywania testów, należy ocenić ich wpływ na testy, które już zostały wykonane.

Podczas implementacji testów analityk testów musi potwierdzić dostępność konkretnych osób odpowiedzialnych za przygotowanie i utrzymanie środowiska testowego, dostępność wszystkich testaliów oraz gotowość narzędzi testowych i powiązanych procesów do użycia. Dotyczy to zarządzania konfiguracją, zarządzania defektami oraz logowania wyników testów i zarządzania testami. Ponadto analityk testów musi zweryfikować procedury gromadzenia danych wykorzystywanych do oceny bieżącego statusu w odniesieniu do kryteriów wyjścia oraz do raportowania rezultatów testów.

Przy implementacji testów warto zastosować zrównoważone podejście oparte na ustaleniach z fazy planowania testów. Na przykład analityczną strategię testową opartą na ryzyku często łączy się z reaktywną strategią testową. W takich sytuacjach pewną część implementacji testów stanowi przygotowanie testów (tzw. testów swobodnych), w których nie postępuje się zgodnie z wcześniej zdefiniowanymi skryptami.

Testy swobodne (ang. *unscripted*) nie powinny być przypadkowe ani pozbawione określonego celu, gdyż wtedy trudno określić ich czas trwania i uzyskane pokrycie, a to wiąże się z niewielką liczbą wykrywanych defektów. Należy je przeprowadzać w sesjach o określonych ramach czasowych i ustalić wstępny przebieg za pomocą karty opisu testu, jednak trzeba zachować możliwość odejścia od zaleceń podanych w karcie,

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

jeśli w trakcie sesji zostaną zidentyfikowane potencjalnie bardziej produktywne możliwości testowania. Na przestrzeni lat praktycy opracowali szereg technik testowania opartych na doświadczeniu, takich jak ataki usterek [Whittaker03], zgadywanie błędów [Myers11] i testowanie eksploracyjne [Whittaker09]. Takie podejścia nie eliminują analizy, projektowania i implementacji testów, ale czynności te są realizowane w dużej mierze podczas wykonywania testów.

Jeśli stosowane są takie reaktywne strategie testowe, rezultaty każdego testu mają wpływ na analizę, projektowanie i implementację kolejnych testów. Strategie tego rodzaju co prawda wymagają mniejszego nakładu pracy i często umożliwiają efektywne wykrywanie defektów, mają jednak również wady, między innymi takie jak:

- wymagają od analityka testów wiedzy specjalistycznej,
- trudno przewidzieć czas ich trwania,
- trudno sprawdzić uzyskane pokrycie,
- ich powtarzalność może być bardzo ograniczona bez wsparcia solidnej dokumentacji lub wsparcia ze strony narzędzi.

1.6. Wykonywanie testów

Testy wykonywane są zgodnie z harmonogramem wykonywania testów. Obejmuje to następujące działania (patrz dokument [ISTQB_FL_SYL]):

- wykonywanie testów manualnych, w tym testowanie eksploracyjne,
- wykonywanie testów automatycznych,
- porównywanie rzeczywistych rezultatów z oczekiwanymi,
- analizowanie anomalii w celu ustalenia ich prawdopodobnych przyczyn,
- zgłaszanie defektów na podstawie zaobserwowanych awarii,
- rejestrowanie rzeczywistych rezultatów wykonania testów,
- aktualizacja powiązań między podstawą testów a testaliami, z uwzględnieniem rezultatów testów,
- wykonywanie testów regresji.

Wymienione powyżej zadania może realizować zarówno tester, jak i analityk testów.

Poniżej przedstawiono listę typowych zadań dodatkowych, które mogą być wykonywane przez analityka testów:

- identyfikowanie skupisk defektów, które mogą oznaczać konieczność dodatkowego przetestowania pewnego fragmentu przedmiotu testów,
- zgłaszanie propozycji przyszłych sesji testowania eksploracyjnego na podstawie odkryć z dotychczasowych testów eksploracyjnych,
- identyfikowanie nowych czynników ryzyka na podstawie informacji uzyskanych podczas realizowania zadań związanych z wykonywaniem testów,
- zgłaszanie propozycji udoskonalenia wybranych produktów pracy z etapu implementacji testów (np. poprawienia procedur testowych).

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

2. Zadania analityka testów w testowaniu opartym na ryzyku — 60 minut

Słowa kluczowe
identyfikacja ryzyka, łagodzenie ryzyka, ryzyko produktowe, testowanie oparte na ryzyku

Cele nauczania związane z zadaniami analityka testów w testowaniu opartym na ryzyku

Zadania analityka testów w testowaniu opartym na ryzyku

TA-2.1.1. (K3) Kandydat potrafi zidentyfikować czynniki ryzyka dla podanej sytuacji, dokonać oceny ryzyka i zaproponować odpowiednie środki łagodzenia ryzyka.

© International Software Testing Qualifications Board
Stowarzyszenie Jakości Systemów Informatycznych

2.1. Wprowadzenie

Kierownik testów często jest całkowicie odpowiedzialny za ustanowienie oraz zarządzanie strategią testów opartą na ryzyku. Kierownik testów zwykle angażuje analityka testów w prace nad zapewnieniem poprawnej implementacji podejścia opartego na ryzyku.

Analityk testów powinien być czynnie zaangażowany w następujące czynności związane z testowaniem opartym na ryzyku:
- identyfikację ryzyka,
- ocenę ryzyka,
- łagodzenie ryzyka.

Te zadania są wykonywane iteracyjnie na przestrzeni całego cyklu wytwarzania oprogramowania, tak aby zespół projektowy mógł reagować na pojawiające się czynniki ryzyka i zmianę ich priorytetów oraz regularnie oceniać status ryzyka i informować o nim interesariuszy (dodatkowe informacje można znaleźć w publikacjach [vanVeenendaal12] i [Black02]). W zwinnym wytwarzaniu oprogramowania trzy wspomniane czynności są często łączone w ramach tzw. sesji zarządzania ryzykiem, które koncentrują się na danej iteracji lub wydaniu.

Analityk testów powinien działać w ramach struktury testowania opartego na ryzyku zbudowanej przez kierownika testów dla potrzeb danego projektu. Powinien przy tym korzystać ze swojej znajomości czynników ryzyka w danej dziedzinie biznesowej, które mogą pojawić się w projekcie, takich jak czynniki ryzyka związane z bezpieczeństwem, kwestiami biznesowymi i ekonomicznymi oraz czynnikami politycznymi.

2.2. Identyfikacja ryzyka

W procesie identyfikacji ryzyka szanse na wykrycie jak największej liczby potencjalnych istotnych czynników ryzyka są tym większe, im większa grupa interesariuszy weźmie w nim udział.

Analitycy testów często dysponują unikatową wiedzą dotyczącą dziedziny biznesowej testowanego systemu, dlatego są szczególnie predysponowani do wykonywania pewnych zadań, takich jak:
- prowadzenie wywiadów z ekspertami z danej dziedziny i użytkownikami,
- dokonywanie samodzielnych ocen,
- korzystanie z szablonów ryzyka,
- udział w warsztatach dotyczących ryzyka,
- udział w sesjach „burzy mózgów” z obecnymi i potencjalnymi użytkownikami,
- definiowanie list kontrolnych do testowania,
- korzystanie z wcześniejszych doświadczeń z podobnymi systemami lub projektami.

Analityk testów powinien przede wszystkim ściśle współpracować z użytkownikami i innymi ekspertami z danej dziedziny (np. specjalistami w dziedzinie inżynierii wymagań i analitykami biznesowymi), aby ustalić, które obszary ryzyka biznesowego powinny zostać uwzględnione podczas testowania. W zwinnym wytwarzaniu oprogramowania, dzięki bliskim kontaktom z interesariuszami, można regularnie prowadzić działania związane z identyfikacją ryzyka, np. w trakcie spotkań dotyczących planowania iteracji.

Przykłady czynników ryzyka, które mogą zostać zidentyfikowane w projekcie, to:
- problemy z poprawnością funkcjonalną, np. niepoprawne obliczenia,
- problemy z użytecznością, np. brak potrzebnych skrótów klawiszowych,
- problemy z przenaszalnością, np. brak możliwości zainstalowania aplikacji na określonych platformach.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

2.3. Ocena ryzyka

Podczas gdy identyfikacja ryzyka polega na wskazaniu jak największej liczby istotnych ryzyk, ocena ryzyka jest analizą tych zidentyfikowanych ryzyk. W szczególności obejmuje ona kategoryzację każdego ryzyka oraz określenie poziomu tegoż ryzyka.

Określenie poziomu ryzyka obejmuje z reguły ocenę (dla każdego elementu ryzyka) jego prawdopodobieństwa i wpływu. Prawdopodobieństwo ryzyka zwykle jest rozumiane jako prawdopodobieństwo, że dany, potencjalny problem, istnieje w testowanym systemie i zostanie zaobserwowany w działaniu produkcyjnym tego systemu. Wkład technicznego analityka testów powinien polegać na wyszukiwaniu czynników ryzyka i określaniu prawdopodobieństwa ich wystąpienia, a wkład analityka testów — na ocenie potencjalnego wpływu biznesowego wystąpienia danego problemu (w zwinnym wytwarzaniu oprogramowania takie rozróżnienie pomiędzy rolami może być mniej ścisłe).

Wpływ ryzyka jest często rozumiany jako wielkość szkody wpływającej na użytkowników, klientów lub innych interesariuszy. Wywodzi się on zatem z ryzyka biznesowego. Wkład analityka testów powinien polegać na identyfikowaniu dziedziny biznesowej oraz ocenianiu potencjalnego wpływu poszczególnych czynników ryzyka na użytkowników. Do czynników wpływających na ryzyko biznesowe należą m. in.:

- częstotliwość używania dotkniętej ryzykiem funkcji,
- straty biznesowe,
- straty finansowe,
- potencjalne straty lub obciążenia środowiskowe albo społeczne,
- konsekwencje prawne, cywilne lub karne,
- kwestie związane z bezpieczeństwem,
- grzywny, utrata licencji,
- brak uzasadnionych technicznie i ekonomicznie sposobów ominięcia problemu w przypadku braku możliwości kontynuowania pracy,
- widoczność funkcji,
- widoczność awarii, prowadząca do negatywnego rozgłosu i potencjalnej utraty reputacji,
- utrata klientów.

Na podstawie dostępnych informacji o ryzyku analityk testów wyznacza poziom ryzyka biznesowego według wytycznych określonych przez kierownika testów. Poziomy mogą być określone przy wykorzystaniu skali porządkowej (np. liczb rzeczywistych, poziomów: niski/średni/wysoki, kolorów sygnalizacji świetlnej). Od momentu, gdy prawdopodobieństwo ryzyka i wpływ ryzyka zostały wyznaczone, kierownik testów korzysta z tych wartości w celu określenia poziomu ryzyka dla każdego elementu ryzyka. Ten poziom ryzyka jest następnie wykorzystywany do priorytetyzacji czynności łagodzenia ryzyka [vanVeenendaal 12].

2.4. Łagodzenie ryzyka

W czasie trwania projektu analityk testów powinien stawiać sobie następujące cele:

- Zmniejszenie ryzyka produktowego poprzez zastosowanie dobrze zaprojektowanych przypadków testowych, które jednoznacznie wykazują, czy testy zostały zaliczone czy niezaliczone, oraz poprzez uczestnictwo w przeglądach produktów pracy związanych z oprogramowaniem, takich jak wymagania, projekty i dokumentacja dla użytkowników.
- Podejmowanie właściwych działań łagodzących ryzyko wskazanych w strategii testów i planie testów (np. testowanie procesu biznesowego związanego ze szczególnie dużym ryzykiem, z zastosowaniem konkretnych technik testowania).
- Ponowna ocena znanych czynników ryzyka w oparciu o dodatkowe informacje zgromadzone w toku projektu, korygowanie wielkości prawdopodobieństwa ryzyka, wpływu ryzyka lub obu tych wartości.
- Identyfikowanie nowych czynników ryzyka na podstawie informacji zgromadzonych podczas testowania.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

Testowanie ma podstawowy wkład w łagodzenie ryzyka produktowego. Poprzez wykrywanie defektów testerzy zmniejszają ryzyko, informując o istnieniu wykrytych defektów i umożliwiając ich usunięcie przed wydaniem oprogramowania. Jeżeli testerzy nie znajdują żadnych defektów, testowanie zmniejsza ryzyko poprzez zapewnienie, że w pewnych warunkach (warunkach, w jakich wykonano testy) system działa prawidłowo. Analityk testów pomaga określić możliwości łagodzenia ryzyka m.in. poprzez badanie możliwości gromadzenia dokładnych danych testowych, tworzenie i weryfikowanie realistycznych scenariuszy użycia oraz prowadzenie lub nadzorowanie badań użyteczności.

2.4.1. Ustalanie priorytetów testów

Poziom ryzyka jest również wykorzystywany do ustalania priorytetów testów. Analityk testów może na przykład ustalić, że istnieje wysokie ryzyko w obszarze dokładności transakcji w systemie księgowym. Aby złagodzić to ryzyko, tester może podjąć współpracę z ekspertami biznesowymi i zgromadzić zestaw dobrych danych przykładowych, które można przetworzyć w celu weryfikacji dokładności wyników przetwarzania. Analityk testów może także stwierdzić, że poważne ryzyko w nowym produkcie stanowią problemy z użytecznością. Zamiast czekać na wykrycie problemów podczas testów akceptacyjnych wykonywanych przez użytkowników, analityk testów może przeprowadzić wczesny test użyteczności o wysokim priorytecie z wykorzystaniem prototypu, co pozwoli zidentyfikować i usunąć problemy jeszcze przed rozpoczęciem testów akceptacyjnych. Priorytety testów należy rozważyć we wczesnych fazach planowania, tak, aby dopasować harmonogram do priorytetów testów i wykonać niezbędne testy w odpowiednim momencie.

W niektórych przypadkach wszystkie testy czynników wysokiego ryzyka są wykonywane przed testami czynników niższego ryzyka, a wykonywanie odbywa się w kolejności ściśle związanej z oceną ryzyka (podejście „w głąb” (ang. *depth-first*)); w innych sytuacjach wybiera się próbkę testów reprezentujących wszystkie zidentyfikowane obszary ryzyka, ważoną według wartości poziomu ryzyka, aby zapewnić pokrycie każdego czynnika przynajmniej jednym testem (podejście „wszerz” (ang. *breadth-first*)).

Bez względu na to, czy testowanie oparte na ryzyku odbywa się według podejścia "w głąb" czy "wszerz", czas przeznaczony na testowanie może nie wystarczyć na wykonanie wszystkich testów. Testowanie oparte na ryzyku umożliwia testerom złożenie kierownictwu raportu podsumowującego pozostały w danym momencie poziom ryzyka, a kierownictwu — podjęcie decyzji, czy przedłużyć testy, czy też przenieść pozostałe ryzyko na użytkowników, klientów, personel wsparcia technicznego i/lub personel operacyjny.

2.4.2. Dostosowywanie testów na potrzeby przyszłych cykli testowania

Ocena ryzyka nie jest jednorazowym działaniem wykonywanym przed rozpoczęciem implementacji testów, ale stanowi ciągły proces. Każdy planowany w przyszłości cykl testowy powinien wiązać się z nową analizą ryzyka, aby uwzględnić takie czynniki jak:

- dowolne nowe lub istotnie zmienione czynniki ryzyka produktowego,
- niestabilne lub podatne na awarie obszary systemu wykryte podczas testowania,
- czynniki ryzyka związane z usuniętymi defektami,
- typowe defekty wykrywane podczas testowania,
- słabo przetestowane obszary (słabo pokryte wymaganiami).

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

3. Techniki testowania — 630 minut

Słowa kluczowe
analiza wartości brzegowych, czarnoskrzynkowa technika testowania, karta opisu testu, podział na klasy równoważności, taksonomia defektów, technika drzewa klasyfikacji, technika testowania oparta na defektach, technika testowania oparta na doświadczeniu, testowanie eksploracyjne, testowanie w oparciu o doświadczenie, testowanie przejść pomiędzy stanami, testowanie sposobem par, testowanie w oparciu o listę kontrolną, testowanie w oparciu o tablicę decyzyjną, zgadywanie błędów

Cele nauczania — techniki testowania

3.1. Wprowadzenie
Nie określono celów nauczania.

3.2. Czarnoskrzynkowe techniki testowania

TA-3.2.1. (K4) Kandydat potrafi przeanalizować podane elementy specyfikacji i zaprojektować przypadki testowe korzystając z techniki podziału na klasy równoważności.
TA-3.2.2. (K4) Kandydat potrafi przeanalizować podane elementy specyfikacji i zaprojektować przypadki testowe korzystając z techniki analizy wartości brzegowych.
TA-3.2.3. (K4) Kandydat potrafi przeanalizować podane elementy specyfikacji i zaprojektować przypadki testowe korzystając z techniki testowania w oparciu o tablicę decyzyjną.
TA-3.2.4. (K4) Kandydat potrafi przeanalizować podane elementy specyfikacji i zaprojektować przypadki testowe korzystając z techniki testowania przejść pomiędzy stanami.
TA-3.2.5. (K2) Kandydat potrafi omówić rolę diagramów drzewa klasyfikacji wspierających techniki testowania.
TA-3.2.6. (K4) Kandydat potrafi przeanalizować podane elementy specyfikacji i zaprojektować przypadki testowe korzystając z techniki testowania sposobem par.
TA-3.2.7. (K4) Kandydat potrafi przeanalizować podane elementy specyfikacji i zaprojektować przypadki testowe korzystając z techniki testowania opartego na przypadkach użycia.
TA-3.2.8. (K4) Kandydat potrafi przeanalizować system (lub jego specyfikację wymagań) w celu ustalenia, jakie typy defektów zostaną w nim prawdopodobnie znalezione i potrafi wybrać odpowiednie czarnoskrzynkowe techniki testowania.

3.3 Techniki testowania oparte na doświadczeniu

TA-3.3.1. (K2) Kandydat potrafi opisać zasady technik testowania opartych na doświadczeniu oraz wskazać ich wady i zalety w porównaniu z technikami czarnoskrzynkowymi oraz technikami opartymi na defektach.
TA-3.3.2. (K3) Kandydat potrafi zidentyfikować testy eksploracyjne dla danego scenariusza.
TA-3.3.3. (K2) Kandydat potrafi opisać zastosowanie technik testowania opartych na defektach i wskazać różnice między ich zastosowaniem a zastosowaniem technik czarnoskrzynkowych.

3.4. Zastosowanie najbardziej odpowiednich technik testowania

TA-3.4.1. (K4) Kandydat potrafi określić, dla podanej sytuacji projektowej, jakie czarnoskrzynkowe lub oparte na doświadczeniu techniki testowania należy zastosować, aby osiągnąć konkretne cele.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

3.1. Wprowadzenie

Techniki testowania opisane w tym rozdziale dzielą się na następujące kategorie:
- czarnoskrzynkowe,
- oparte na doświadczeniu.

Obie kategorie technik uzupełniają się i mogą być stosowane jako odpowiednie dla dowolnej, podanej czynności testowej, w testowaniu na dowolnym poziomie testów, w zależności od potrzeb.

Należy zauważyć, że oba rodzaje technik można stosować w testowaniu zarówno funkcjonalnych, jak i niefunkcjonalnych charakterystyk jakościowych. Testowanie charakterystyk oprogramowania zostało opisane w następnym rozdziale.

Techniki testowania opisane w poniższych podrozdziałach mają za cel ustalenie optymalnych danych testowych (np. na podstawie klas równoważności) lub wyprowadzanie procedur testowych (np. na podstawie modeli stanów). Z reguły przy tworzeniu kompletnych przypadków testowych łączy się ze sobą różne techniki.

3.2. Czarnoskrzynkowe techniki testowania

Czarnoskrzynkowe techniki testowania zostały przedstawione w sylabusie ISTQB® poziomu podstawowego [ISTQB_FL_SYL].

Najważniejsze wspólne cechy czarnoskrzynkowych technik testowania:
- podczas projektowania testów są tworzone modele zgodne z techniką testowania, np. diagramy przejść pomiędzy stanami, tablice decyzyjne itp.,
- warunki testowe są systematycznie wyprowadzane z tych modeli.

Techniki testowania określają zwykle kryteria pokrycia, za pomocą których można mierzyć czynności związane z projektowaniem i wykonywaniem testów. Całkowite spełnienie kryteriów pokrycia nie oznacza, że cały zestaw testowy jest kompletny, a jedynie to, że z danego modelu nie można już wywieść więcej testów umożliwiających zwiększenie pokrycia w oparciu o przyjętą technikę.

Testowanie czarnoskrzynkowe zwykle opiera się na dokumentacji zawierającej specyfikację rozwiązania, np. specyfikację wymagań lub historyjki użytkownika. Dokumentacja taka powinna określać zachowanie systemu, zwłaszcza w zakresie funkcjonalności, zatem zwykle elementem procesu testowania zachowania systemu jest wyprowadzanie testów z wymagań. W pewnych sytuacjach dokumentacja specyfikacji może nie być dostępna, ale istnieją wymagania, które można wywnioskować, takie jak zastąpienie funkcjonalności wcześniejszego systemu.

Istnieje szereg czarnoskrzynkowych technik testowania. Techniki te są przeznaczone do różnych rodzajów oprogramowania i scenariuszy zastosowania. Poniżej opisano możliwości zastosowania poszczególnych technik wraz z ograniczeniami i trudnościami, na jakie może się natknąć analityk testów, metody pomiaru pokrycia oraz typy defektów, na jakie ukierunkowana jest dana technika.

Dodatkowe informacje można znaleźć w publikacjach [ISO29119-4], [Bath14], [Beizer95], [Black07], [Black09], [Copeland04], [Craig02], [Forgács19], [Koomen06], [Myers11] i [Roman18].

3.2.1. Podział na klasy równoważności

Technika podziału na klasy równoważności jest używana do zmniejszenia liczby przypadków testowych wymaganych do efektywnego przetestowania obsługi danych wejściowych, wyjściowych, wartości wewnętrznych i wartości uwarunkowanych czasowo. W procesie podziału powstają klasy równoważności, czyli zbiory wartości, które powinny być przetwarzane w taki sam sposób. Przyjmuje się, że w przypadku

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

wybrania jednej reprezentatywnej wartości z danej klasy jest zapewnione pokrycie wszystkich elementów tej klasy.

Zazwyczaj zachowanie przedmiotu testów determinuje nie jeden, a kilka parametrów. W przypadku łączenia w ramach przypadków testowych klas równoważności zdefiniowanych dla kilku parametrów mogą być stosowane różne techniki.

Obszar zastosowania

Tę technikę można stosować na każdym poziomie testów w sytuacjach, gdy wszystkie elementy zbioru wartości do przetestowania mają zostać przetworzone w taki sam sposób i gdy zbiory wartości używanych przez aplikację nie wchodzą ze sobą w interakcję. Klasą równoważności może być dowolny niepusty zbiór wartości, w szczególności może to być zbiór uporządkowany, nieuporządkowany, dyskretny, ciągły, nieskończony, skończony lub nawet zbiór jednoelementowy. Dobór zbiorów wartości dotyczy zarówno poprawnych, jak i niepoprawnych klas (tzn. klas zawierających wartości, które są uznawane za niepoprawne w przypadku testowanego oprogramowania).

Ta technika sprawdza się najlepiej w połączeniu z analizą wartości brzegowych, która rozszerza listę testowanych wartości o wartości brzegowe klas równoważności. Podział na klasy równoważności i wykorzystanie wartości z poprawnych klas to typowa technika używana w testach dymnych nowej wersji lub nowego wydania oprogramowania, ponieważ umożliwia szybkie ustalenie, czy działają podstawowe funkcje.

Ograniczenia/trudności

Jeżeli założenia dotyczące równoważności są niepoprawne i nie wszystkie wartości w poszczególnych zbiorach są przetwarzane w taki sam sposób, zastosowanie tej techniki może nie wystarczyć do wychwycenia defektów. Ważne jest również staranne dobieranie klas. Na przykład pole formularza, w którym podaje się liczby ujemne lub dodatnie, mogłoby być przetestowane lepiej dla dwóch klas danych poprawnych: osobno dla liczb dodatnich i ujemnych, ze względu na możliwe odmienne przetwarzanie. W zależności od tego, czy wartość „zero” jest dozwolona, czy nie, może ona stać się kolejną klasą równoważności. Aby dobrać najlepsze klasy równoważności, analityk testów powinien rozumieć przetwarzanie związane z przedmiotem testów. Może to oznaczać konieczność uzyskania wyjaśnień dotyczących struktury kodu.

Analityk testów powinien również wziąć pod uwagę możliwe zależności pomiędzy klasami równoważności dla różnych parametrów. Na przykład, w systemie rezerwacji lotów parametr „opiekun” może być użyty jedynie w kombinacji z klasą „dziecko” dla parametru określającego wiek pasażera.

Pokrycie

Pokrycie ustala się poprzez podzielenie liczby klas, dla których przetestowano przynajmniej jedną wartość, przez liczbę wszystkich zidentyfikowanych klas. Pokrycie klas równoważności podaje się jako wartość procentową. Użycie kilku wartości z jednej klasy nie zwiększa pokrycia w ujęciu procentowym.

Jeśli zachowanie przedmiotu testów zależy od jednego parametru, każda klasa równoważności, zarówno poprawna jak i niepoprawna, powinna być pokryta przynajmniej raz.

W przypadku więcej niż jednego parametru analityk testów powinien wybrać proste lub kombinatoryczne pokrycie, zależnie od poziomu ryzyka [Offutt16]. Istotne jest zatem rozróżnienie pomiędzy kombinacjami zawierającymi jedynie wartości z klas poprawnych, a kombinacjami zawierającymi jedną lub więcej wartości z klas niepoprawnych. W przypadku kombinacji zawierających wartości jedynie z klas poprawnych minimalnym wymaganiem jest tzw. proste pokrycie każdej z poprawnych klas równoważności dla każdego parametru. Minimalna liczba przypadków testowych potrzebnych w tym przypadku jest równa największej liczbie poprawnych klas równoważności spośród wszystkich rozważanych parametrów, zakładając, że wartości parametrów są od siebie niezależne. Bardziej dokładne typy pokrycia związane z technikami kombinatorycznymi obejmują m.in. pokrycie dla testowania sposobem par (zob. punkt 3.2.6. poniżej) lub

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

pełne pokrycie wszystkich możliwych kombinacji poprawnych klas równoważności. Niepoprawne klasy równoważności powinny być przetestowane co najmniej indywidualnie, tzn. w kombinacji z wyłącznie poprawnymi klasami równoważności dla pozostałych parametrów, w celu uniknięcia zjawiska maskowania defektów. Zatem w przypadku prostego pokrycia, każda niepoprawna klasa równoważności wnosi jeden dodatkowy przypadek testowy do zbioru testów. W przypadku wysokiego ryzyka do zbioru testów można dodać inne kombinacje, np. złożone wyłącznie z niepoprawnych klas równoważności lub z par niepoprawnych klas równoważności.

Typy defektów

Analityk testów używa tej techniki do wykrywania defektów związanych z obsługą różnych wartości danych.

3.2.2. Analiza wartości brzegowych

Technika analizy wartości brzegowych służy do testowania poprawności obsługi wartości znajdujących się na granicach uporządkowanych klas równoważności. Dwa najpopularniejsze sposoby zastosowania tej techniki to dwupunktowa i trójpunktowa analiza wartości brzegowych. W przypadku testowania metodą dwupunktową używa się wartości brzegowej i wartości sąsiedniej, wykraczającej poza daną klasę (z najmniejszym możliwym przyrostem wynikającym z założonego poziomu wymaganej dokładności). Na przykład, jeżeli klasa równoważności dla wartości waluty (a więc z dokładnością do dwóch miejsc po przecinku) zawiera wartości od 1 do 10, to testy dla górnej wartości brzegowej tej klasy w technice dwupunktowej uwzględniają wartości 10 i 10,01. Z kolei wartości dla dolnej wartości brzegowej tej klasy testy uwzględniają wartości 1 i 0,99. Wartości brzegowe danej klasy są określone przez maksymalną i minimalną wartość zdefiniowanej klasy równoważności.

W przypadku testowania metodą trójpunktową używa się wartości tuż poniżej wartości brzegowej, samej wartości brzegowej i wartości tuż powyżej niej. W poprzednim przykładzie test dla górnej wartości brzegowej rozważanej klasy uwzględniałby wartości 9,99, 10 i 10,01, a dla dolnej zostałyby przetestowane wartości 1,01, 1 i 0,99. Wybór między testowaniem metodą dwupunktową a trójpunktową powinien być podyktowany ryzykiem związanym z testowanym elementem, przy czym podejścia trójpunktowego używa się w przypadku elementów o większym ryzyku.

Obszar zastosowania

Ta technika ma zastosowanie na wszystkich poziomach testów i można jej użyć, gdy istnieją uporządkowane klasy równoważności. Z tego powodu jest często stosowana wraz z techniką podziału na klasy równoważności. Klasy równoważności muszą być uporządkowane ze względu na konieczność użycia elementów zbioru leżących na jego brzegu i tuż poza nim. Na przykład zakres liczb jest klasą uporządkowaną. Klasa zawierająca pewne ciągi znaków może być uporządkowana np. poprzez wprowadzenie porządku leksykograficznego, ale jeśli uporządkowanie nie jest istotne z biznesowego lub technicznego punktu widzenia, stosowanie analizy wartości brzegowych nie wnosi wartości dodanej. Oprócz zakresów liczb, inne przykłady klas równoważności, do których można zastosować analizę wartości brzegowych, to:

- atrybuty liczbowe zmiennych innych niż liczbowe (np. długość),
- liczba wykonań iteracji pętli, w tym pętli w diagramach przejść stanów,
- liczba elementów iteracji przechowywanych w strukturach danych takich jak tablice,
- rozmiar obiektów fizycznych (w tym pamięci),
- czas trwania czynności.

Ograniczenia/trudności

Dokładność tej techniki zależy od właściwego zidentyfikowania granic klas równoważności, czyli wyznaczenia samych klas. Wiąże się to zatem z takimi samymi ograniczeniami i trudnościami, jak w przypadku podziału na klasy równoważności. Analityk testów powinien również zdawać sobie sprawę ze ścisłości (dokładności) dla wartości poprawnych i niepoprawnych, aby móc ustalić odpowiednie wartości do przetestowania. Analizę wartości brzegowych można wykorzystać tylko z klasami uporządkowanymi, ale nie ogranicza się to wyłącznie do zakresów wartości poprawnych. Na przykład w przypadku testowania

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

liczby komórek obsługiwanych w arkuszu kalkulacyjnym bierze się pod uwagę klasę zawierającą liczbę komórek do maksymalnej liczby dozwolonych komórek włącznie (wartość brzegowa) oraz klasę rozpoczynającą się od liczby komórek o jeden większej od liczby maksymalnej (ponad wartością brzegową).

Pokrycie

Pokrycie ustala się poprzez podzielenie liczby przetestowanych wartości brzegowych przez liczbę wszystkich zidentyfikowanych wartości brzegowych[1] (zarówno dla testowania dwu- jak i trójpunktowego). Pokrycie jest wyrażane w procentach.
Podobnie jak w przypadku klas równoważności, w przypadku wielu parametrów analityk testów powinien wybrać prosty lub kombinatoryczny typ pokrycia, w zależności od ryzyka.

Typy defektów

Analiza wartości brzegowych sprawdza się przy wyszukiwaniu błędnie zdefiniowanych lub identyfikowaniu niezdefiniowanych granic zbiorów, a także umożliwia wykrycie nadmiarowych granic. Za pomocą tej techniki można wykryć defekty związane z przetwarzaniem wartości brzegowych, zwłaszcza błędy użycia operatorów „mniejsze niż" i „większe niż” (np. zamiana). Można jej też użyć do wykrywania defektów niefunkcjonalnych, np. związanych z limitami (system obsługujący 10 000 jednoczesnych użytkowników, ale nie 10 001).

3.2.3. Testowanie w oparciu o tablicę decyzyjną

Tablica decyzyjna jest tabelaryczną reprezentacją zbioru warunków i związanych z nimi akcji. Zbiór ten wyraża reguły wskazujące, które akcje powinny wystąpić dla określonego zbioru zachodzących warunków [OMG-DMN]. Analityk testów może skorzystać z tablic decyzyjnych do analizy reguł opisujących testowane oprogramowanie i zaprojektować testy pokrywające te reguły.

Warunki i wynikające z nich akcje przedmiotu testów tworzą wiersze tablicy decyzyjnej, przy czym zazwyczaj warunki znajdują się w górnej, a akcje – w dolnej części tablicy. Pierwsza kolumna tablicy zawiera odpowiednio opisy warunków i akcji. Kolejne kolumny, zwane regułami, zawierają wartości warunków i odpowiadające im wartości akcji.

Tablice decyzyjne, w których warunki przyjmują jedynie dwie wartości logiczne „Prawda” i „Fałsz”, są zwane tablicami decyzyjnymi z ograniczonym zakresem wejść. Przykładem takiego warunku może być warunek „Dochód klienta < 1 000”. Tablice decyzyjne z rozszerzonym zakresem wejść pozwalają, aby warunki przyjmowały również inne niż logiczne wartości. Na przykład, warunek „Dochód klienta” może przyjąć jedną z trzech możliwych wartości: „mniejszy niż 1 000”, „pomiędzy 1 000 a 2 000”, „większy niż 2 000”.

Proste akcje przyjmują wartości logiczne „Prawda” i „Fałsz” (np. akcja „Przyznana zniżka = 20%” przyjmuje wartość „Prawda” zazwyczaj oznaczaną w tablicy decyzyjnej znakiem „X”, jeśli akcja powinna zajść, albo wartość „Fałsz” oznaczaną zazwyczaj znakiem „–”, jeśli akcja nie powinna zajść). Tak jak w przypadku warunków, akcje mogą również przyjmować wartości z innych dziedzin. Na przykład akcja „Przyznana zniżka” może przyjąć jedną z pięciu możliwych wartości: 0%, 10%, 20%, 35% i 50%.

Testowanie oparte na tablicach decyzyjnych rozpoczyna się od zaprojektowania tablic decyzyjnych na podstawie specyfikacji. Reguły zawierające nieosiągalne kombinacje warunków są usuwane lub oznaczane jako „nieosiągalne”. Następnie analityk testów powinien wraz z innymi interesariuszami dokonać przeglądu zaprojektowanych tablic decyzyjnych. Analityk testów powinien się upewnić, że reguły w tablicy są spójne (tzn. reguły nie nachodzą na siebie), pełne (tzn. zawierają reguły dla każdej osiągalnej kombinacji wartości warunków) oraz poprawne (tzn. modelują w sposób właściwy pożądane zachowanie).

[1] W przypadku techniki trójpunktowej nie wszystkie wartości muszą być wartościami brzegowymi (np. w omawianym wcześniej przykładzie dla wartości brzegowej 10 do testów brane są wartości 9.99, 10 i 10.01, gdzie wartość 9.99 nie jest brzegiem żadnej klasy), ale są wliczane do metryki, gdyż stanowią elementy pokrycia. [przyp. tłum.]

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

Podstawowa zasada w testowaniu opartym na tablicach decyzyjnych polega na tym, że reguły odpowiadają warunkom testowym.
Projektując przypadek testowy dla określonej reguły, analityk testów powinien mieć świadomość, że dane wejściowe dla przypadku testowego mogą być innymi parametrami niż w warunkach tabeli decyzyjnej. Na przykład wartość "TRUE" warunku "wiek ≥ 18?" może wymagać od testera obliczenia wieku na podstawie parametrów wejściowych data urodzenia i bieżąca data. Podobnie oczekiwane wyniki przypadku testowego mogą być pośrednimi konsekwencjami działań tabeli decyzyjnej.

Gdy tablica decyzyjna jest gotowa, reguły powinny być zaimplementowane w postaci przypadków testowych poprzez odpowiedni wybór wartości wejściowych i oczekiwanych wyników, które spełnią odpowiednie warunki i akcje.

Zminimalizowane tablice decyzyjne

Próba przetestowania wszystkich możliwych kombinacji wejść dla odpowiadających im warunków może prowadzić do powstania ogromnych tablic decyzyjnych. Pełna tablica decyzyjna z ograniczonym zakresem wejść dla n warunków posiada 2^n reguł. Technika polegająca na systematycznym zmniejszaniu liczby kombinacji nosi nazwę testowania w oparciu o zminimalizowane tablice decyzyjne [Copeland04, Roman18]. Przy użyciu tej techniki grupa reguł z tym samym zestawem akcji może być zredukowana (zminimalizowana) do jednej reguły jeśli, w ramach tej grupy, niektóre warunki są nieistotne dla akcji, a wszystkie pozostałe warunki nie zmieniają swoich wartości. W tak powstałej regule wartości warunków nieistotnych oznaczane są jako „nieistotne”, zazwyczaj poprzez wykorzystanie symbolu myślnika „–”. Dla warunków nieistotnych analityk testów może zdefiniować dowolne poprawne wartości przy implementacji testów.

Z innym przypadkiem minimalizowania reguł mamy do czynienia w sytuacji, gdy wartość warunku nie może współistnieć w kombinacji z jakąś wartością innego warunku lub gdy wartości dwóch bądź więcej warunków są sprzeczne (taką wartość oznacza się w tablicy jako „nie dotyczy”). Na przykład, w tablicy decyzyjnej dla płatności kartą, jeśli warunek „karta jest poprawna” jest fałszywy, warunek „kod PIN jest poprawny” nie może być zastosowany.

Zminimalizowane tablice decyzyjne mogą mieć o wiele mniej reguł niż pełne tablice, co skutkuje mniejszą liczbą przypadków testowych i mniejszym wysiłkiem testowym. Jeśli reguła posiada wartość „nieistotne” i regułę tę pokrywa tylko jeden przypadek testowy, tylko jedna z kilku możliwych wartości odpowiedniego warunku będzie przetestowana dla tej reguły, więc defekt związany z wystąpieniem innej wartości może nie zostać wykryty. Dlatego w przypadku wysokiego poziomu ryzyka analityk testów w porozumieniu z kierownikiem testów powinien zdefiniować osobne reguły dla każdej osiągalnej kombinacji pojedynczych wartości warunków, zamiast minimalizować tablicę.

Obszar zastosowania

Testowanie w oparciu o tablice decyzyjne stosuje się z reguły na poziomach testów integracyjnych, systemowych i akceptacyjnych. Może ono się sprawdzać również w testowaniu modułowym, jeżeli dany moduł zawiera logikę decyzyjną. Technika testowania w oparciu o tablice decyzyjne jest szczególnie przydatna, gdy przedmiot testów jest opisany w postaci diagramów przepływu lub tablic reguł biznesowych.

Tablice decyzyjne są również stosowane jako technika definiowania wymagań i czasami specyfikacji wymagań, które mogą już być zdefiniowane w takiej postaci. Analityk testów powinien nadal brać udział w przeglądzie tablic decyzyjnych i analizować je przed rozpoczęciem projektowania testów.

Ograniczenia/trudności

Jeśli rozpatrujemy kombinacje warunków, ustalenie wszystkich warunków wchodzących ze sobą w interakcje może być trudne, zwłaszcza wtedy, gdy wymagania nie są dobrze zdefiniowane lub nie zostały w ogóle udokumentowane. Podczas definiowania tablicy decyzyjnej trzeba pamiętać o wyborze

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

odpowiednich warunków, tak, aby liczba kombinacji była również ograniczona. W najgorszym przypadku liczba reguł będzie wzrastała wykładniczo.

Pokrycie

Najczęściej stosowany standard pokrycia dla tej techniki wymaga, aby pokryć każdą regułę tablicy decyzyjnej jednym przypadkiem testowym. Pokrycie jest mierzone jako liczba reguł objętych zestawem testowym podzielona przez całkowitą liczbę wykonalnych reguł, wyrażone w procentach.

Techniki analizy wartości brzegowych i klas równoważności uzupełniają technikę testowania w oparciu o tablice decyzyjne, zwłaszcza w przypadku tablic z rozszerzonym zakresem wejść. Jeśli wartości warunków są uporządkowanymi klasami równoważności, analiza wartości brzegowych może być użyta w celu zidentyfikowania dodatkowych wartości prowadzących do dodatkowych reguł i przypadków testowych.

Typy defektów

Typowe defekty to nieprawidłowe przetwarzanie logiki biznesowej przy określonych kombinacjach warunków, zakończone nieoczekiwanymi rezultatami. Podczas tworzenia tablic decyzyjnych mogą zostać wykryte defekty w dokumencie specyfikacji. Często okazuje się, że oczekiwany rezultat dla przygotowanego zestawu warunków nie jest określony (dla co najmniej jednej reguły). Najczęstsze defekty to pominięcia akcji (tzn. brak informacji, co powinno się zdarzyć w określonej sytuacji) oraz sprzeczności.

3.2.4. Testowanie przejść pomiędzy stanami

Testowanie przejść pomiędzy stanami umożliwia weryfikację zdolności przedmiotu testów do wchodzenia w zdefiniowane stany i wychodzenia z nich poprzez poprawne przejścia, a także próbę wchodzenia w niepoprawne stany oraz pokrywania niepoprawnych przejść. W reakcji na zdarzenia przedmiot testów przechodzi ze stanu w (inny lub ten sam) stan i wykonuje działania. Zdarzenia mogą mieć dodatkowy kwalifikator warunku (nazywany czasem warunkiem dozoru albo warunkiem sprawdzającym przejścia), który wpływa na wybór ścieżki przejścia. Na przykład w wyniku zdarzenia logowania przy użyciu poprawnej kombinacji nazwy użytkownika i hasła zostanie wykonane inne przejście niż w przypadku zdarzenia logowania z podanym niepoprawnym hasłem. Informacje takie przedstawiane są w postaci diagramu stanów albo tablicy stanów (która może również zawierać potencjalne niepoprawne przejścia między stanami [Roman18]).

Obszar zastosowania

Testowanie przejść pomiędzy stanami można zastosować w przypadku każdego oprogramowania o zdefiniowanych stanach, w którym występują zdarzenia powodujące przejścia pomiędzy tymi stanami (np. przejścia na inny ekran aplikacji). Testowanie przejść pomiędzy stanami sprawdza się na wszystkich poziomach testowania. Dobrymi kandydatami do tego rodzaju testów są: oprogramowanie wbudowane, aplikacje WWW i systemy transakcyjne, a także systemy sterowania, np. sterowniki sygnalizatorów świetlnych.

Ograniczenia/trudności

Najtrudniejszą częścią definiowania tablicy lub diagramu stanów jest zwykle ustalenie listy stanów. W przypadku przedmiotu testów z interfejsem użytkownika stany często reprezentują ekrany wyświetlane przez aplikację. W oprogramowaniu wbudowanym stany mogą zależeć od stanów elementów sprzętowych.

Oprócz stanów podstawową jednostką testowania w tej technice jest pojedyncze przejście. Przy prostym przetestowaniu wszystkich pojedynczych przejść można wykryć pewne defekty przejść pomiędzy stanami, ale większe możliwości daje testowanie sekwencji przejść. Pojedyncze przejście nazywane jest 0-przełączeniem. Sekwencję dwóch następujących po sobie przejść określa się jako 1-przełączenie. Sekwencję trzech kolejnych przejść – jako 2-przełączenie itd. W ogólności, N-przełączenie reprezentuje sekwencję N+1 kolejnych przejść [Chow1978]. Wraz ze wzrostem N bardzo szybko wzrasta również liczba

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

N-przełączeń, co utrudnia osiągnięcie pokrycia N-przełączeń za pomocą rozsądnego, małego zbioru testów.

Pokrycie

Podobnie jak w przypadku innych technik testowania istnieje tu hierarchia poziomów pokrycia. Minimalny, akceptowalny stopień pokrycia, obejmuje odwiedzenie wszystkich stanów i wykonanie wszystkich przejść co najmniej jednokrotnie. Stuprocentowe pokrycie przejść (nazywane też 100% pokryciem 0-przełączeń) gwarantuje, że każdy stan został odwiedzony i każde przejście wykonane, chyba że projekt systemu lub model przejść pomiędzy stanami (diagram albo tablica) zawierają defekty. W zależności od relacji między stanami a przejściami może być konieczne wielokrotne wykonanie niektórych przejść, aby wykonać inne przejścia na zależnych od nich ścieżkach.

Pojęcie „pokrycie N-przełączeń" odnosi się do liczby wykonanych przejść długości N+1 i wyrażane jest jako wartość procentowa (odsetek wszystkich przejść tej długości). Na przykład w celu osiągnięcia 100% pokrycia 1-przełączeń należy co najmniej raz przetestować każdą poprawną sekwencję dwóch następujących po sobie przejść. W tego rodzaju teście można wykryć pewne typy awarii, które nie ujawniają się przy 100% pokrycia 0-przełączeń.

Pojęcie „pokrycie okrążenia" (ang. *round-trip*) odnosi się do sytuacji, gdy sekwencje przejść tworzą pętle. Stuprocentowe pokrycie okrążenia jest osiągane, gdy przetestowano wszystkie pętle z dowolnego stanu prowadzące do tego samego stanu, dla wszystkich stanów, w których takie pętle zaczynają się i kończą. Pętle mogą zawierać co najwyżej jedno wystąpienie dowolnego stanu (z wyjątkiem stanów początkowego/końcowego, które są tożsame) [Offutt16].

Niezależnie od przyjętego podejścia można uzyskać jeszcze wyższą wartość pokrycia poprzez próbę uwzględnienia przejść niepoprawnych zidentyfikowanych w tablicy przejść pomiędzy stanami. Wymagania dotyczące pokrycia używane do testowania przejść pomiędzy stanami muszą zawierać informację, czy uwzględniono w nich przejścia niepoprawne.

Diagram stanów lub tablica stanów dla konkretnego przedmiotu testów ułatwiają projektowanie przypadków testowych i uzyskanie zakładanego pokrycia. Informacje takie można również przedstawić w postaci tablicy zawierającej N-przełączenia dla konkretnej wartości N [Roman18], [Black09].

Identyfikację elementów uwzględnianych w pokryciu (np. przejść, stanów lub N-przełączeń) można również przeprowadzić w ramach procedury manualnej. Sugerowaną metodą działania jest wydrukowanie diagramu stanów lub tablicy stanów i zaznaczanie uwzględnionych elementów do momentu uzyskania zakładanego pokrycia [Black09]. Jeśli diagramy stanów lub tablice stanów są bardziej złożone, takie postępowanie może okazać się zbyt czasochłonne. Dlatego do obsługi testowania przejść pomiędzy stanami należy użyć odpowiedniego narzędzia.

Typy defektów

Typowe defekty wykrywane w testach tego typu to między innymi (zob. także [Beizer95]):

- niepoprawne typy lub wartości zdarzeń,
- niepoprawne typy lub wartości akcji,
- niepoprawny stan początkowy,
- brak możliwości osiągnięcia pewnych stanów końcowych,
- brak możliwości wejścia do określonych stanów,
- dodatkowe (niepotrzebne, nadmiarowe) stany,
- brak możliwości wykonania pewnych poprawnych przejść,
- możliwość wykonania niepoprawnych przejść,
- niepoprawne warunki dozoru.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

Podczas tworzenia modelu przejść pomiędzy stanami w specyfikacji wymagań mogą zostać wykryte defekty. Najczęstsze defekty to pominięcia (tzn. brak informacji, co powinno się zdarzyć w określonej sytuacji) oraz sprzeczności.

3.2.5. Technika drzewa klasyfikacji

Drzewa klasyfikacji są wykorzystywane w niektórych czarnoskrzynkowych technikach testowania. Stanowią graficzną reprezentację tworzonej przestrzeni danych związanej z przedmiotem testów.

Dane są zorganizowane w ramach klasyfikacji i klas w następujący sposób:

- Klasyfikacje: reprezentują parametry w przestrzeni danych przedmiotu testów, takie jak parametry wejściowe (zawierające stany środowiska i warunki wstępne) czy parametry wyjściowe. Na przykład jeśli aplikacja może być skonfigurowana na wiele sposobów, klasyfikacje mogą uwzględniać klienta, przeglądarkę, język i system operacyjny.
- Klasy: każda klasyfikacja może mieć dowolną liczbę klas i podklas opisujących wystąpienia parametru. Każda klasa (lub klasa równoważności) opisuje pewną wartość w ramach klasyfikacji. W powyższym przykładzie klasyfikacja języka może zawierać klasy równoważności odpowiadające językowi angielskiemu, francuskiemu i hiszpańskiemu.

Drzewa klasyfikacji pozwalają analitykom testów wprowadzać potrzebne kombinacje. Obejmuje to na przykład kombinacje par (patrz punkt 3.2.6.), kombinacje trójek wartości i pojedyncze wartości.

Dodatkowe informacje na temat sposobów korzystania z techniki drzewa klasyfikacji można znaleźć w publikacjach [Bath14], [Black09] i [Roman18].

Obszar zastosowania

Dzięki utworzeniu drzewa klasyfikacji analityk testów może zidentyfikować parametry (klasyfikacje) i ich klasy równoważności (klasy), które warto uwzględnić w testowaniu.

Dalsza analiza diagramu drzewa klasyfikacji umożliwia określenie potencjalnych wartości brzegowych i pewnych kombinacji danych wejściowych, które są szczególnie interesujące albo które należy pominąć (np. z powodu ich niezgodności). Wyjściowe drzewo klasyfikacji może zostać następnie wykorzystane w dzieleniu na klasy równoważności, analizie wartości brzegowych lub w testowaniu sposobem par (patrz punkt 3.2.6.).

Ograniczenia/trudności

Gdy liczba klasyfikacji i/lub klas rośnie, diagram staje się bardziej rozbudowany i trudniej się nim posługiwać. Ponadto, technika drzewa klasyfikacji nie tworzy kompletnych przypadków testowych, a jedynie kombinacje danych testowych. Do obowiązków analityków testów należy tworzenie pełnych przypadków testowych na podstawie wyników tych kombinacji.

Pokrycie

Przypadki testowe można na przykład projektować z myślą o minimalnym pokryciu klas (tj. co najmniej jednokrotnym przetestowaniu wszystkich wartości w ramach klasyfikacji). Analityk testów może także podjąć decyzję o pokryciu kombinacji par lub użyć innego podejścia kombinatorycznego, np. pokrycia kombinacji trójek wartości.

Typy defektów

Typy wykrywanych defektów zależą od techniki, którą wspierają drzewa klasyfikacji (podział na klasy równoważności, analiza wartości brzegowych lub testowanie sposobem par).

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

3.2.6. Testowanie sposobem par

Testowanie sposobem par stosuje się w przypadku oprogramowania, dla którego należy sprawdzić kombinacje wielu parametrów wejściowych, z których każdy przyjmuje wiele różnych wartości. Liczba wszystkich kombinacji jest wtedy zbyt duża, aby było możliwe ich przetestowanie w przewidzianym czasie. Parametry wejściowe mogą być niezależne, tj. każda opcja każdego czynnika (dowolnie wybrana wartość dowolnego parametru wejściowego) może utworzyć kombinację z każdą opcją dowolnego innego czynnika, choć nie zawsze tak być musi (zob. poniżej uwagę o modelach cech). Kombinacja konkretnego parametru (zmiennej lub czynnika) i konkretnej wartości tego parametru nosi nazwę pary parametr-wartość. Na przykład jeśli istnieje parametr „kolor", który może przybierać jedną z siedmiu dozwolonych wartości, w tym kolor czerwony, to parą parametr-wartość mogłaby być para „kolor = czerwony".

W testowaniu sposobem par wykorzystywane są techniki kombinatoryczne, aby zapewnić co najmniej jednokrotne przetestowanie każdej pary parametr-wartość danego parametru w połączeniu z każdą parą parametr-wartość każdego innego parametru (innymi słowy, żeby zostały przetestowane wszystkie pary par parametr-wartość dla każdych dwóch różnych parametrów), a jednocześnie aby uniknąć testowania wszystkich kombinacji par parametr-wartość. Jeśli analityk testów stosuje podejście manualne, tworzy tablicę, której wiersze reprezentują przypadki testowe, a każda kolumna odpowiada jednemu parametrowi. Następnie analityk testów wypełnia ją wartościami w taki sposób, aby znalazły się w niej wszystkie możliwe pary wartości (patrz [Roman18]). Wszystkie puste wpisy w tablicy analityk testów może wypełnić wartościami zgodnie z wiedzą na temat danej dziedziny.

Istnieje wiele narzędzi wspomagających analityka testów w tym zadaniu (przykłady są dostępne pod adresem www.pairwise.org). W takich narzędziach należy wprowadzić listę parametrów i ich wartości, aby wygenerować odpowiedni zbiór kombinacji wartości parametrów pokrywających wszystkie pary par parametr-wartość. Dane wyjściowe z narzędzia można wykorzystać jako dane wejściowe dla przypadków testowych. Należy pamiętać, że analityk testów powinien określić oczekiwane rezultaty dla każdej kombinacji utworzonej w narzędziach.

Drzewa klasyfikacji (patrz punkt 3.2.5.) są często stosowane w połączeniu z testowaniem sposobem par [Bath14]. Istnieją narzędzia wspomagające projektowanie drzewa klasyfikacji i umożliwiające wizualizację kombinacji parametrów i ich wartości (w niektórych narzędziach dostępne są opcje definiowania par). Dzięki temu można określić następujące informacje:

- Wartości wejściowe używane w technice testowania sposobem par.
- Konkretne interesujące kombinacje (np. często używane wartości lub częste źródła defektów).
- Konkretne kombinacje niezgodne. Nie oznacza to, że czynniki połączone w kombinacji nie będą na siebie wzajemnie wpływać; może się tak zdarzyć, ale ten wpływ powinien mieć akceptowalny zakres.
- Powiązania logiczne między zmiennymi. Przykład: „jeśli zmienna1 = x, to zmienna2 nie może przyjmować wartości y". Drzewa klasyfikacji, w których zapisano takie informacje o powiązaniach, są nazywane modelami cech (ang. *feature models*).

Obszar zastosowania

Problem zbyt wielu kombinacji wartości parametrów przejawia się w co najmniej dwóch sytuacjach związanych z testowaniem. W niektórych elementach testowych występuje kilka parametrów, każdy z kilkoma możliwymi wartościami — na przykład ekran z kilkoma polami do wprowadzania danych. Kombinacje wartości parametrów stanowią dane wejściowe dla takich przypadków testowych. Ponadto niektóre systemy można konfigurować w kilku różnych wymiarach, co skutkuje potencjalnie dużą przestrzenią możliwych konfiguracji. W obu tych sytuacjach można posłużyć się techniką testowania sposobem par, aby wybrać odpowiedni i osiągalny podzbiór kombinacji, który jest zarządzalny i możliwy do wykonania.

W przypadku parametrów o dużej liczbie wartości można najpierw zastosować technikę podziału na klasy równoważności lub inny mechanizm wyboru, aby zmniejszyć liczbę wartości danego parametru, a następnie

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

wykonać testowanie sposobem par, aby zmniejszyć zbiór kombinacji wynikowych. Zapisanie parametrów i ich wartości w drzewie klasyfikacji ułatwia realizację tego zadania.

Opisane techniki stosuje się zwykle na poziomach testów integracji modułów, testów systemowych i testów integracji systemów.

Ograniczenia/trudności

Głównym ograniczeniem opisywanych technik jest założenie, że rezultaty kilku testów są reprezentatywne dla wszystkich testów, a testy te reprezentują oczekiwane sposoby użycia systemu. Jeżeli dana kombinacja nie zostanie wybrana do przetestowania, nieoczekiwane interakcje między pewnymi zmiennymi mogą pozostać ukryte. Ponadto te techniki trudno wyjaśnić odbiorcom bez przygotowania technicznego, ponieważ mogą oni mieć trudności ze zrozumieniem logiki redukcji liczby testów. Warto przy okazji poinformować zainteresowanych o wynikach badań empirycznych [Kuhn16]. W objętych nimi urządzeniach medycznych 66% awarii było związane z jedną zmienną, natomiast zsumowany wskaźnik awarii wywołanych przez jedną zmienną lub interakcję dwóch zmiennych wyniósł 97%. Istnieje rezydualne ryzyko związane z możliwością braku wykrycia w testach sposobem par awarii systemu wynikających z interakcji co najmniej trzech zmiennych.

Czasem trudności sprawia identyfikacja parametrów i ich odpowiednich wartości. Dlatego zadanie to powinno być realizowane z wykorzystaniem drzew klasyfikacji, jeśli jest to możliwe (patrz sekcja 3.2.5.). Trudno ręcznie znaleźć minimalny zestaw kombinacji, który zapewni określony poziom pokrycia, dlatego do znalezienia możliwie małego zbioru kombinacji z reguły używa się specjalnych narzędzi. Niektóre narzędzia potrafią wymusić uwzględnienie lub wykluczenie pewnych kombinacji w ostatecznym zbiorze. Analityk testów może posłużyć się tą funkcją, aby położyć większy lub mniejszy nacisk na pewne czynniki, zgodnie ze swoją wiedzą z danej dziedziny lub informacjami o sposobach użytkowania produktu.

Pokrycie

W stuprocentowym pokryciu sposobem par każda para wartości dowolnej pary różnych parametrów musi być uwzględniona w co najmniej jednej kombinacji.

Typy defektów

Defekty wykrywane najczęściej przy użyciu tej techniki testowania wiążą się z kombinacjami warunków dwóch parametrów.

3.2.7. Testowanie oparte na przypadkach użycia

Testowanie oparte na przypadkach użycia umożliwia przeprowadzenie transakcyjnych testów opartych na doświadczeniu, które powinny naśladować użytkowanie modułu lub systemu opisane w przypadkach użycia. Przypadki użycia definiują interakcje aktorów z modułem lub systemem służące osiągnięciu jakiegoś celu. Aktorami mogą być użytkownicy (ludzie), zewnętrzny sprzęt oraz inne moduły lub systemy.

Powszechnie przyjęty standard dla przypadków użycia jest opisany w [OMG-UML].

Obszar zastosowania

Testowanie oparte na przypadkach użycia wykonuje się zwykle na poziomie testów systemowych i akceptacyjnych. Jeśli zachowanie modułów lub systemów zostało opisane w formie przypadków użycia, ten rodzaj testowania można też zastosować w testach integracyjnych. Przypadki użycia często stanowią również podstawę testów wydajnościowych, ponieważ odzwierciedlają realistyczne użytkowanie systemu. Zachowania opisane w przypadkach użycia można przypisać do użytkowników wirtualnych w celu zbudowania realistycznego obciążenia systemu (o ile w samych scenariuszach albo dla tych scenariuszy określono wymagania dotyczące obciążenia i wydajności).

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

Ograniczenia/trudności

Aby przypadki użycia były poprawne, muszą opisywać realistyczne działania użytkownika. Specyfikacje przypadków użycia są formą projektu systemu. Wymagania opisujące cele, które mają zrealizować użytkownicy, powinny pochodzić od samych użytkowników lub ich przedstawicieli. Przed zaprojektowaniem odpowiednich przypadków użycia należy zweryfikować takie wymagania, porównując je z wymaganiami organizacyjnymi. Wartość przypadku użycia jest ograniczona, gdy nie odzwierciedla on rzeczywistych wymagań użytkowników i wymagań organizacyjnych. Taki przypadek nie ułatwia realizacji zadań użytkowników, a raczej ją utrudnia.

Aby pokrycie było dokładne, należy poprawnie zdefiniować wyjątki, ścieżki alternatywne i obsługę błędów. Przypadki użycia powinny służyć jako wskazówki, ale nie jako kompletna definicja elementów do przetestowania, ponieważ mogą nie zawierać jasnej definicji całego zestawu wymagań. Warto utworzyć na podstawie opisu słownego przypadku użycia również inne modele, na przykład diagramy przepływu sterowania (ang. *flowcharts*) i/lub tablice decyzyjne, aby zwiększyć dokładność testowania i zweryfikować sam przypadek użycia. Podobnie jak w wypadku innych rodzajów specyfikacji, taki proces ułatwia wykrycie ewentualnych anomalii logicznych w opisie przypadków użycia, o ile one istnieją.

Pokrycie

Minimalny dopuszczalny poziom pokrycia przypadku użycia to jeden przypadek testowy do obsługi podstawowego zachowania i wystarczająca liczba dodatkowych przypadków testowych odpowiadających wszystkim ścieżkom alternatywnym i obsłudze błędów. Jeśli wymagany jest minimalny zestaw testowy, to wiele różnych alternatywnych zachowań można uwzględnić w ramach jednego przypadku testowego, o ile są one ze sobą zgodne. Jeśli wymagane są bardziej rozbudowane opcje diagnostyczne (np. pomoc w lokalizowaniu defektów), można zaprojektować po jednym dodatkowym przypadku testowym na każdą ścieżkę alternatywną, chociaż może się zdarzyć, że część zagnieżdżonych ścieżek alternatywnych będzie musiała trafić do tych samych przypadków testowych (np. zakończenie lub brak zakończenia działania w ramach obsługi wyjątku „ponów próbę”).

Typy defektów

Wykrywane defekty to: nieprawidłowa obsługa zdefiniowanych zachowań, brak obsługi ścieżek alternatywnych, nieprawidłowe przetwarzanie podanych warunków oraz niepoprawnie zaimplementowane lub nieprawidłowe komunikaty o awariach.

3.2.8. Łączenie technik

Czasem w celu utworzenia przypadków testowych łączy się ze sobą kilka technik. Na przykład warunki zidentyfikowane za pomocą tablicy decyzyjnej można podzielić na klasy równoważności, aby odkryć różne sposoby spełnienia danego warunku. Przypadki testowe pokrywają wówczas nie tylko wszystkie kombinacje warunków, ale również, dla warunków podzielonych na klasy, powinny być zaprojektowane dodatkowe przypadki testowe, aby pokryć dodatkowo poszczególne klasy równoważności. Podejmując decyzję o zastosowaniu danej techniki, analityk testów powinien wziąć pod uwagę jej obszar zastosowania, ograniczenia i trudności z nią związane oraz cele testowania (pokrycie i rodzaj poszukiwanych defektów). Aspekty te zostały przedstawione w opisach poszczególnych technik w niniejszym rozdziale. Może być tak, że w danej sytuacji nie da się wybrać „najlepszej” techniki. Łączenie odpowiednich technik będzie często najskuteczniejszym sposobem osiągnięcia wyznaczonych celów testowych, o ile istnieje wystarczająca ilość czasu i umiejętności, aby prawidłowo zastosować techniki.

3.3. Techniki testowania oparte na doświadczeniu

Testowanie oparte na doświadczeniu wykorzystuje umiejętności i intuicję testerów oraz ich doświadczenia w pracy z aplikacjami i technologiami podobnymi do testowanych. Celem jest takie ukierunkowanie testów, aby zwiększyć skuteczność wykrywania defektów. Techniki testowania tego typu obejmują działania od „szybkich testów”, w których nie przewiduje się formalnie określonych działań do wykonania przez testera,

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

poprzez wstępnie zaplanowane sesje testowe, aż po udokumentowane sesje testowe wykorzystujące karty opisu testów. Takie metody testowania niemal zawsze okazują się przydatne, jednak szczególnej wartości nabierają w pewnych sytuacjach w projekcie.

Zalety testowania opartego na doświadczeniu:

- Może skutecznie zastąpić bardziej strukturalne podejścia w przypadku, gdy brak dokumentacji systemu.
- Można zastosować takie techniki, gdy na testowanie przeznaczono bardzo niewiele czasu.
- W trakcie testowania można skorzystać z wiedzy specjalistycznej związanej z daną dziedziną i technologią. Mogą jej dostarczyć osoby nie biorące udziału w testowaniu, np. analitycy biznesowi i klienci.
- Programiści mogą na wczesnych etapach pracy uzyskać informacje zwrotne.
- Zespół ma okazję poznać oprogramowanie w trakcie jego wytwarzania.
- Jest skuteczne, gdy awarie są analizowane w trakcie wykonania programu.
- Możliwe jest zastosowanie różnorodnych technik testowania.

Wady testowania opartego na doświadczeniu:

- Testowanie tego typu może być niewłaściwe, jeśli wymagana jest szczegółowa dokumentacja testowa.
- Trudno uzyskać wysoki poziom powtarzalności testów.
- Możliwości precyzyjnej oceny pokrycia są ograniczone.
- Testy tego typu trudniej poddają się późniejszej automatyzacji.

W podejściach reaktywnych i heurystycznych testerzy wykonują zwykle testy oparte na doświadczeniu, a testowanie można bardziej elastycznie dopasować do zachodzących zdarzeń niż w przypadku podejść ze ścisłym planowaniem testów. Ponadto wykonanie testów i ocena ich rezultatów odbywają się równocześnie. Niektóre bardziej systematyczne podejścia do testowania opartego na doświadczeniu nie są w pełni dynamiczne, tj. testy nie są tworzone na bieżąco podczas ich wykonywania przez testera. Może tak być np. w sytuacji, gdy przed wykonaniem testów do analizy pewnych aspektów przedmiotu testów stosuje się technikę zgadywania błędów.

Należy zauważyć, że co prawda istnieją pewne wskazówki co do szacowania pokrycia produktu testami, które przedstawiono poniżej, jednak w technikach opartych na doświadczeniu nie istnieją formalne kryteria pokrycia.

3.3.1 Zgadywanie błędów

W technice zgadywania błędów analityk testów korzysta ze swojego doświadczenia, zgadując, jakie potencjalne błędy mogły zostać popełnione podczas projektowania i tworzenia kodu. Po zidentyfikowaniu oczekiwanych błędów analityk testów określa najlepsze metody wykrywania wynikających z nich defektów. Na przykład jeżeli analityk testów spodziewa się, że w oprogramowaniu będą występować awarie w przypadku wprowadzenia niepoprawnego hasła, to przeprowadzi testy polegające na wprowadzaniu różnego rodzaju wartości w polu hasła w celu sprawdzenia, czy rzeczywiście popełniono taki błąd i czy skutkuje on defektem powodującym awarię po uruchomieniu testów.

Zgadywanie błędów jest nie tylko techniką testowania, ale może również być przydatne podczas analizy ryzyka w celu zidentyfikowania potencjalnych trybów awarii [Myers11].

Obszar zastosowania

Technikę zgadywania błędów stosuje się przede wszystkim podczas testów integracyjnych i systemowych, można to jednak robić na dowolnym poziomie testów. Ta technika często jest stosowana razem z innymi technikami i ułatwia poszerzenie zakresu istniejących przypadków testowych. Zgadywanie błędów może się także okazać efektywne podczas testowania nowej wersji oprogramowania pod kątem

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

często występujących defektów jeszcze przed rozpoczęciem bardziej rygorystycznego testowania skryptowego (ang. *scripted testing*).

Ograniczenia/trudności

Ze zgadywaniem błędów wiążą się pewne ograniczenia i trudności:

- Pokrycie jest trudne do oszacowania i w dużej mierze zależy od umiejętności i doświadczenia analityka testów.
- Tę technikę powinni stosować przede wszystkim doświadczeni testerzy, zaznajomieni z typami defektów często spotykanymi w kodzie takiego samego rodzaju, jak ten testowany.
- Zgadywanie błędów jest szeroko stosowane, ale często nie podlega dokumentowaniu, może więc być mniej powtarzalne niż inne formy testowania.
- Zdarza się, że przypadki testowe są dokumentowane w sposób zrozumiały wyłącznie dla autora testów, który jako jedyny potrafi je wykonać ponownie.

Pokrycie

W przypadku wykorzystania taksonomii defektów pokrycie ustala się jako wskaźnik procentowy odpowiadający stosunkowi liczby przetestowanych elementów taksonomii do łącznej liczby elementów taksonomii. Jeżeli nie zastosowano żadnej taksonomii defektów, pokrycie jest ograniczone doświadczeniem i wiedzą testera oraz ilością dostępnego czasu. Liczba znalezionych za pomocą tej techniki defektów bywa różna w zależności od umiejętności testera wynajdowania problematycznych obszarów.

Typy defektów

Zazwyczaj wykrywane są defekty zdefiniowane w wybranej taksonomii defektów lub „odgadnięte” przez analityka testów, które mogą pozostać niewykryte przez testy czarnoskrzynkowe.

3.3.2. Testowanie w oparciu o listę kontrolną

W technice testowania w oparciu o listę kontrolną doświadczony analityk testów korzysta z wysokopoziomowej, ogólnej listy elementów, na które należy zwrócić uwagę, które należy sprawdzić lub o których należy pamiętać, albo z zestawu reguł czy kryteriów, pod kątem których trzeba sprawdzić przedmiot testów. Listy kontrolne są tworzone w oparciu o standardy, doświadczenie i inne źródła. Na przykład, lista kontrolna standardu interfejsu użytkownika może być wykorzystana jako podstawa do testowania aplikacji. W projektach zwinnych listy kontrolne można budować na podstawie kryteriów akceptacji historyjki użytkownika.

Obszar zastosowania

Testowanie w oparciu o listę kontrolną najlepiej sprawdza się w projektach z doświadczonym zespołem testowym, zaznajomionym z testowanym oprogramowaniem lub obszarem, którego dotyczy lista kontrolna (np. aby skutecznie posłużyć się listą kontrolną dla interfejsu użytkownika, analityk testów nie musi znać konkretnego, testowanego systemu, wystarczy znajomość problematyki testowania interfejsów użytkownika). Listy kontrolne mają postać wysokopoziomową i nie uwzględniają szczegółowych kroków charakterystycznych dla przypadków testowych i procedur testowych, luki są więc uzupełniane przez testera na podstawie własnej wiedzy. Dzięki pominięciu szczegółowych instrukcji listy kontrolne nie wymagają dużych nakładów pielęgnacji i można je stosować w wielu podobnych wydaniach oprogramowania.

Listy kontrolne sprawdzają się w sytuacji, gdy wersje oprogramowania pojawiają się często, a modyfikacje są szybko wprowadzane. Pozwala to skrócić czas poświęcany na przygotowania i utrzymanie dokumentacji testowej. Listy kontrolne można wykorzystywać na wszystkich poziomach testowania, a także w testach regresji i testach dymnych.

Ograniczenia/trudności

Wysokopoziomowy charakter list kontrolnych może wpływać na powtarzalność rezultatów testów. Różni testerzy mogą różnie interpretować listę kontrolną i weryfikować poszczególne elementy różnymi metodami. Mogą w ten sposób uzyskać różne rezultaty testów mimo wykorzystania tej samej listy kontrolnej. Zwiększa

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

to potencjalnie pokrycie, ale czasem cierpi na tym powtarzalność. Listy kontrolne mogą również w nieuzasadniony sposób zwiększać zaufanie względem rzeczywistego poziomu pokrycia, ponieważ przebieg testów zależy od oceny danego testera. Listy kontrolne można budować na podstawie bardziej szczegółowych przypadków testowych lub innych list. Zwykle w miarę upływu czasu zwiększa się ich długość. Wymagana jest pielęgnacja list, tak, aby ich zawartość pokrywała istotne aspekty testowanego oprogramowania.

Pokrycie

Pokrycie ustala się jako wskaźnik procentowy odpowiadający stosunkowi liczby przetestowanych elementów listy kontrolnej do łącznej liczby elementów tej listy. Pokrycie jest tak dobre, jak dobra jest lista kontrolna, ale ze względu na jej wysokopoziomowy charakter rzeczywisty wynik zależy od analityka testów, który wykonuje test.

Typy defektów

Najczęściej wykrywane za pomocą tej techniki defekty powodują awarie wynikające z wprowadzania różnych danych, wykonywania kroków w różnej kolejności lub indywidualnego modyfikowania przebiegu pracy podczas testowania.

3.3.3. Testowanie eksploracyjne

Podczas testowania eksploracyjnego tester jednocześnie poznaje przedmiot testów i jego defekty, planuje działania testowe do wykonania, projektuje i wykonuje testy oraz raportuje ich rezultaty. Tester dynamicznie dopasowuje cele testowania podczas jego wykonywania i opracowuje tylko minimum dokumentacji [Whittaker09].

Obszar zastosowania

Dobrze przeprowadzone testowanie eksploracyjne jest zaplanowane, wykonywane interaktywnie i twórczo. Technika ta nie wymaga obszernej dokumentacji testowanego systemu, dlatego często się z niej korzysta w sytuacjach, gdy taka dokumentacja nie jest dostępna lub nie nadaje się do wykorzystania z innymi technikami testowania. Testowanie eksploracyjne jest często stosowane jako uzupełnienie innych rodzajów testów i jako podstawa do opracowywania dodatkowych przypadków testowych. Testowanie eksploracyjne jest często stosowane w zwinnym wytwarzaniu oprogramowania w celu szybkiego, adaptacyjnego przetestowania historyjek użytkownika przy minimalnych wymaganiach dotyczących dokumentacji. Technikę tę można jednak również zastosować w projektach opartych na sekwencyjnych modelach wytwarzania.

Ograniczenia/trudności

Testy eksploracyjne nie zapewniają żadnej miary pokrycia. Ponadto odtworzenie przeprowadzonych testów może być trudne. Do technik zarządzania testami eksploracyjnymi należą karty opisu testów, jakie mają być zrealizowane w danej sesji testowej, oraz ramy czasowe (ang. *time-boxing*) w celu określenia czasu przeznaczonego na takie testy. Na zakończenie sesji lub zestawu sesji testowania, kierownik testów może zorganizować spotkanie podsumowujące (ang. *debriefing session*) w celu zebrania rezultatów testów i ustalenia, jakie karty opisu testów będą potrzebne w następnych sesjach testowych.

Innym problemem związanym z sesjami eksploracyjnymi jest ich dokładne śledzenie przy użyciu systemu do zarządzania testami. Czasem tworzy się w tym celu przypadki testowe, które są w istocie sesjami testowania eksploracyjnego. Umożliwia to śledzenie czasu przydzielonego na testy eksploracyjne i zaplanowanego pokrycia wraz z tymi metrykami dla innych rodzajów testowania.

Powtarzalność testów eksploracyjnych może być dość ograniczona, co również może powodować problemy, gdy zajdzie potrzeba odtworzenia kroków, które doprowadziły do awarii. W niektórych organizacjach do rejestracji kroków wykonanych przez testera eksploracyjnego stosuje się funkcję rejestrowania i odtwarzania narzędzia do testów automatycznych. W ten sposób można uzyskać pełny zapis wszystkich czynności wykonanych podczas sesji testowania eksploracyjnego (lub innej sesji testowania

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

opartego na doświadczeniu). Analiza tego zapisu pod kątem rzeczywistego powodu awarii może być uciążliwa, ale stanowi on przynajmniej pewną formę utrwalenia wykonanych kroków.

Do zarejestrowania sesji testowania eksploracyjnego można także użyć innych narzędzi, ale nie rejestrują one oczekiwanych wyników, ponieważ nie zapisują interakcji z graficznym interfejsem użytkownika. W tym wypadku należy zanotować oczekiwane wyniki, tak, aby w razie potrzeby przeprowadzić właściwą analizę defektów. Ogólnie rzecz biorąc, tworzenie notatek w trakcie wykonywania testowania eksploracyjnego jest zalecane, jeśli powtarzalność testów ma znaczenie w projekcie.

Pokrycie

W celu określenia zadań, celów i oczekiwanych produktów prac można zastosować karty opisu testu. Następnie planuje się sesje testowania eksploracyjnego w taki sposób, aby osiągnąć zapisane kryteria. Karta opisu testu może również wskazywać obszary, w których należy skoncentrować testy, elementy należące do zakresu sesji testowania i pozostające poza tym zakresem oraz zasoby potrzebne do wykonania zaplanowanych testów. W ramach sesji testowania można skoncentrować poszukiwania na określonych typach defektów lub innych potencjalnie problematycznych obszarach, które nie wymagają sformalizowania procesu testowania skryptowego.

Typy defektów

Defekty najczęściej wykrywane za pomocą testów eksploracyjnych to problemy scenariuszowe, które nie zostały wychwycone podczas testów funkcjonalności opartych na dokumentacji, problemy, które nie dają się sklasyfikować jako przynależne do konkretnej funkcjonalności oraz problemy związane z przepływem pracy. Podczas testowania eksploracyjnego można również czasem wykryć problemy z wydajnością lub zabezpieczeniami.

3.3.4. Techniki testowania oparte na defektach

W technikach testowania opartych na defektach punktem wyjścia dla projektu testu jest typ poszukiwanych defektów; testy systematycznie wyprowadza się z wiedzy na temat danego typu defektów. W odróżnieniu od testowania czarnoskrzynkowego, w którym testy wyprowadzane są z podstawy testów, w technikach opartych na defektach testy tworzy się na podstawie list związanych z defektami. Listy tego typu mogą być uporządkowane według typów defektów, ich podstawowych przyczyn, objawów awarii i innych danych związanych z defektami. Listy standardowe mają zastosowanie do różnych rodzajów oprogramowania i nie wiążą się z konkretnymi produktami. Pozwala to wykorzystać standardową wiedzę branżową do utworzenia konkretnych testów. Dzięki zgodności z listami branżowymi, metryki występowania defektów można śledzić na poziomie wielu projektów, a nawet organizacji. Najczęściej jednak stosuje się listy defektów specyficzne dla danej organizacji lub projektu, związane z konkretnymi doświadczeniami i wiedzą specjalistyczną.

W testowaniu opartym na defektach można również używać list zidentyfikowanych czynników ryzyka oraz scenariuszy ryzyka. Ta technika testowania umożliwia analitykowi testów ukierunkowanie działań na określony typ defektów lub przejście krok po kroku przez listę opisującą znane i częste defekty danego typu. Na podstawie tych informacji analityk testów tworzy warunki testowe i przypadki testowe, które powodują ujawnienie tych defektów (jeżeli są one obecne w produkcie).

Obszar zastosowania

Testowanie oparte na defektach sprawdza się na każdym poziomie testów, ale z reguły jest stosowane w testach systemowych.

Ograniczenia/trudności

Istnieje wiele taksonomii defektów i niektóre z nich są zorientowane na konkretne typy testów, takie jak testowanie użyteczności. Ważny jest dobór odpowiedniej taksonomii do testowanego oprogramowania (o ile takie taksonomie istnieją). Dla innowacyjnego oprogramowania takie taksonomie mogą nie być dostępne. Niektóre organizacje opracowały własne taksonomie prawdopodobnych lub często spotykanych

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

defektów. Niezależnie od stosowanej taksonomii defektów ważne jest zdefiniowanie oczekiwanego pokrycia przed rozpoczęciem testowania.

Pokrycie

Ta technika udostępnia kryteria pokrycia, na podstawie których można ustalić, czy zidentyfikowano wszystkie przydatne przypadki testowe. W zależności od listy defektów pokrycie może uwzględniać elementy strukturalne, elementy specyfikacji lub scenariusze użytkowe, a także kombinacje tych elementów. W praktyce kryteria pokrycia w technikach opartych na defektach są słabiej usystematyzowane niż w technikach czarnoskrzynkowych — dane są tylko ogólne reguły pokrycia, a decyzje o granicach użytecznego pokrycia są podejmowane na poziomie konkretnych projektów. Podobnie jak w przypadku innych technik, spełnienie kryteriów pokrycia nie oznacza, że zbudowano pełny zestaw testowy, ale że na podstawie rozważanych defektów nie można już opracować w tej technice większej liczby przydatnych testów.

Typy defektów

Typy wykrywanych defektów zależą zwykle od używanej taksonomii defektów. Na przykład jeżeli użyto listy defektów interfejsu użytkownika, większość wykrytych defektów będzie prawdopodobnie związana z interfejsem użytkownika, ale przy okazji tych testów mogą zostać wykryte również inne defekty.

3.4. Zastosowanie najbardziej odpowiedniej techniki

Techniki czarnoskrzynkowe i oparte na doświadczeniu są najbardziej skuteczne, jeśli zostaną zastosowane wspólnie. Techniki testowania oparte na doświadczeniu wypełniają luki w osiągnięciu celów testów wynikające z możliwych systemowych słabości czarnoskrzynkowych technik testowania.

Nie istnieje technika idealna, sprawdzająca się w każdej sytuacji. Analityk testów powinien rozumieć zalety i wady poszczególnych technik i umieć dobrać najbardziej odpowiednią technikę lub zestaw technik do danej sytuacji: typu projektu, harmonogramu, dostępności informacji, umiejętności testera i innych czynników, które mogą mieć wpływ na ten dobór.

Informacje podane w opisie poszczególnych technik czarnoskrzynkowych i technik opartych na doświadczeniu (odpowiednio w podrozdziałach 3.2. i 3.3.), w sekcjach „Obszar zastosowania”, „Ograniczenia/trudności” i „Pokrycie”, stanowią wskazówki dla analityka testów dokonującego wyboru technik najbardziej odpowiednich do zastosowania w danej sytuacji.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

4. Testowanie charakterystyk jakościowych oprogramowania — 180 minut

Słowa kluczowe

adekwatność funkcjonalna, dostępność, doświadczenie użytkownika, estetyka interfejsu użytkownika, funkcjonalność, inwentarz pomiarów użyteczności oprogramowania (SUMI), inwentarz analizy i pomiaru stron internetowych (WAMMI), kompletność funkcjonalna, łatwość nauki, łatwość obsługi, ochrona przed błędami użytkownika, poprawność funkcjonalna, użyteczność, współdziałanie, kompatybilność

Cele nauczania dotyczące testowania charakterystyk jakościowych oprogramowania

4.1. Wprowadzenie

Nie określono celów nauczania.

4.2 Charakterystyki jakościowe w testowaniu w dziedzinie biznesowej

TA-4.2.1. (K2) Kandydat potrafi wskazać techniki testowania, które są właściwe do testowania kompletności funkcjonalnej, poprawności funkcjonalnej i adekwatności funkcjonalnej.

TA-4.2.2. (K2) Kandydat potrafi wymienić typowe defekty, jakie powinny zostać wykryte podczas testowania kompletności funkcjonalnej, poprawności funkcjonalnej i adekwatności funkcjonalnej.

TA-4.2.3. (K2) Kandydat potrafi określić, w którym momencie w cyklu wytwarzania oprogramowania należy testować kompletność funkcjonalną, poprawność funkcjonalną i adekwatność funkcjonalną.

TA-4.2.4. (K2) Kandydat potrafi wyjaśnić podejścia odpowiednie do zweryfikowania i walidacji implementacji wymagań dotyczących użyteczności oraz spełnienia oczekiwań użytkownika.

TA-4.2.5. (K2) Kandydat potrafi wyjaśnić, na czym polega rola analityka testów podczas testowania współdziałania, z uwzględnieniem typów defektów, których należy poszukiwać.

TA-4.2.6. (K2) Kandydat potrafi wyjaśnić, na czym polega rola analityka testów podczas testowania przenaszalności, z uwzględnieniem typów defektów, których należy poszukiwać.

TA-4.2.7. (K4) Dla danego zestawu wymagań kandydat potrafi określić warunki testowe niezbędne do zweryfikowania funkcjonalnych i niefunkcjonalnych charakterystyk jakościowych, które są w zakresie obowiązków analityka testów.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

4.1. Wprowadzenie

W poprzednim rozdziale opisano konkretne techniki, którymi może posłużyć się tester; tematem tego rozdziału jest zastosowanie tych technik do oceny charakterystyk używanych do opisywania jakości aplikacji lub systemów informatycznych.

Przedmiotem niniejszego sylabusa są charakterystyki jakościowe, które podlegają ocenie przez analityka testów. Atrybuty oceniane przez technicznego analityka testów zostały opisane w sylabusie poziomu zaawansowanego dla technicznego analityka testów [CTAL-TTA].

Opis charakterystyk jakościowych produktu bazuje na normie ISO 25010 [ISO25010]. Model jakości oprogramowania ISO określa zestaw charakterystyk jakościowych produktu, z których każda może zawierać charakterystyki podrzędne (podcharakterystyki). Zostały one przedstawione w poniższej tabeli wraz ze wskazaniem, które charakterystyki i podcharakterystyki zostały opisane w sylabusie dla analityków testów, a które w sylabusie dla technicznych analityków testów:

Charakterystyka	Podcharakterystyki	Analityk testów	Techniczny analityk testów
Funkcjonalność	Poprawność funkcjonalna, adekwatność funkcjonalna, kompletność funkcjonalna	X	
Niezawodność	Dojrzałość, tolerowanie usterek, odtwarzalność, osiągalność		X
Użyteczność	Stosowność, łatwość nauki, łatwość obsługi, estetyka interfejsu użytkownika, ochrona przed błędami użytkownika, dostępność	X	
Wydajność	Zachowanie w czasie, wykorzystanie zasobów, przepustowość		X
Pielęgnowalność	Analizowalność, modyfikowalność, testowalność, modułowość, możliwość ponownego wykorzystania		X
Przenaszalność	Zdolność adaptacyjna, instalowalność, zastępowalność	X	X
Zabezpieczenia	Poufność, integralność, niezaprzeczalność, rozliczalność, autentykacja		X
Kompatybilność	Współistnienie		X
	Współdziałanie	X	

Taki podział pracy jest stosowany w odpowiednich sylabusach ISTQB®, jednak w różnych organizacjach może przybierać różne formy.

Aby można było sformułować i udokumentować odpowiednią strategię testów, należy zidentyfikować czynniki ryzyka typowe dla wszystkich charakterystyk jakościowych i podcharakterystyk omówionych w tym rozdziale. Testowanie charakterystyk jakościowych wymaga szczególnie starannego doboru właściwego momentu w cyklu wytwarzania oprogramowania, niezbędnych narzędzi, a także dostępności oprogramowania i dokumentacji do testowania oraz fachowej wiedzy technicznej. Bez odpowiedniej strategii dla każdej z charakterystyk i specyficznych potrzeb związanych z jej testowaniem tester może nie mieć do dyspozycji wystarczającej ilości czasu na odpowiednie zaplanowanie, przygotowanie i wykonanie testów w terminie określonym w harmonogramie [Bath14]. Część tych testów, np. testowanie użyteczności, może wymagać przydzielenia szczególnych zasobów ludzkich, szczegółowego zaplanowania, udostępnienia specjalnych laboratoriów i konkretnych narzędzi, szczególnych umiejętności testerskich

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

oraz, w większości przypadków, dużej ilości czasu. Czasem testy użyteczności mogą być wykonywane przez osobną grupę ekspertów ds. użyteczności (interfejsów użytkownika).

Analityk testów może nie być odpowiedzialny za charakterystyki jakościowe, które wymagają bardziej technicznego podejścia; istotne jednak jest to, aby analityk testów był świadomy innych charakterystyk i rozumiał pokrywające się obszary testowania. Na przykład przedmiot testów, który nie zaliczył testów wydajnościowych, może również nie zaliczyć testów użyteczności, jeżeli działa zbyt wolno, żeby użytkownicy mogli z niego efektywnie korzystać. Podobnie przedmiot testów, w którym występują problemy z niektórymi modułami, nie jest prawdopodobnie gotowy do testów przenaszalności, ponieważ problemy na niższym poziomie będzie trudniej wyizolować w zmienionym środowisku.

4.2. Charakterystyki jakościowe w testowaniu w dziedzinie biznesowej

Głównym obszarem działań analityka testów jest testowanie funkcjonalności. Przedmiotem testów funkcjonalności jest to, „co" robi dany obiekt. Podstawą testów funkcjonalności są z reguły wymagania, specyfikacja, konkretna wiedza z danej dziedziny lub przewidywana potrzeba. Testy funkcjonalne mają różną formę w zależności od poziomu testów, na jakim są wykonywane, a także w zależności od cyklu wytwarzania oprogramowania. Na przykład test funkcjonalny wykonywany podczas testów integracyjnych sprawdza funkcjonalność łączonych poprzez interfejs modułów, które implementują jedną zdefiniowaną funkcję. Na poziomie testów systemowych testy funkcjonalne obejmują sprawdzenie funkcjonalności całego systemu. W systemach systemów testowanie funkcjonalności obejmuje testowanie kompleksowe całości zintegrowanych systemów. W testach funkcjonalności stosuje się szeroki zakres technik testowania (patrz Rozdział 3.).

W zwinnym wytwarzaniu oprogramowania testowanie funkcjonalności zwykle obejmuje następujące elementy:

- testowanie konkretnej funkcjonalności (np. historyjek użytkownika) planowanej do zaimplementowania w danej iteracji,
- testowanie regresji dla wszystkich niezmodyfikowanych funkcji.

Oprócz testów funkcjonalności omawianych w tym podrozdziale w zakresie odpowiedzialności analityka testów znajdują się jeszcze dwie charakterystyki jakościowe, które uznaje się za elementy testów niefunkcjonalnych (ukierunkowanych na sprawdzenie, „jak" przedmiot testów udostępnia funkcjonalność).

4.2.1. Testowanie poprawności funkcjonalnej

Testowanie poprawności funkcjonalnej obejmuje weryfikację zgodności aplikacji z podanymi lub wywnioskowanymi wymaganiami i może obejmować również testowanie dokładności obliczeniowej. W testowaniu poprawności funkcjonalnej stosuje się wiele z technik testowania opisanych w Rozdziale 3. i często używa się specyfikacji lub wcześniejszej wersji systemu jako wyroczni testowej. Testowanie poprawności funkcjonalnej może mieć miejsce na dowolnym poziomie testów i jest ukierunkowane na wykrywanie niepoprawnej obsługi danych lub sytuacji.

4.2.2. Testowanie adekwatności funkcjonalnej

Testowanie adekwatności funkcjonalnej obejmuje ocenę i walidację poziomu dopasowania zestawu funkcji do konkretnych zadań, które te funkcje mają realizować. Testowanie może być oparte na projekcie funkcjonalnym (np. przypadkach użycia i/lub historyjkach użytkownika). Testowanie adekwatności funkcjonalnej z reguły ma miejsce podczas testów systemowych, ale może też odbywać się na późnych etapach testów integracyjnych. Defekty wykryte podczas tych testów wskazują, że system nie zaspokoi potrzeb użytkownika w akceptowalny sposób.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

4.2.3. Testowanie kompletności funkcjonalnej

Celem testowania kompletności funkcjonalnej jest ustalenie pokrycia konkretnych zadań i celów użytkownika przez zaimplementowaną funkcjonalność. Kluczowym elementem umożliwiającym sprawdzenie kompletności funkcjonalnej jest śledzenie powiązań pomiędzy punktami specyfikacji (np. wymaganiami, historyjkami użytkownika i przypadkami użycia), a zaimplementowaną funkcjonalnością (np. funkcją, modułem i przepływem pracy). Pomiar kompletności funkcjonalnej może odbywać się w różny sposób, w zależności od konkretnego poziomu testów i stosowanego cyklu wytwarzania oprogramowania. Na przykład kompletność funkcjonalną w projekcie zwinnym można określić na podstawie zaimplementowanych historyjek użytkownika i funkcji. Z kolei testowanie kompletności funkcjonalnej w testowaniu integracji systemów może skupiać się na pokryciu procesów biznesowych wysokiego poziomu.

Narzędzia do zarządzania testami zwykle obsługują opcje określania kompletności funkcjonalnej, o ile analityk testów rejestruje informacje o powiązaniach między przypadkami testowymi a elementami specyfikacji funkcjonalnej. Jeśli poziom kompletności jest niższy niż oczekiwano, może to oznaczać, że system nie został w pełni zaimplementowany.

4.2.4. Testowanie współdziałania

W testach współdziałania weryfikowana jest wymiana informacji między dwoma lub większą liczbą systemów albo modułów. Sprawdzana jest możliwość wymiany informacji i jej późniejszego wykorzystania. Testy powinny pokrywać wszystkie przewidywane środowiska docelowe (w tym różnice w zakresie sprzętu, oprogramowania, oprogramowania warstwy pośredniej, systemów operacyjnych itp.), aby zapewnić prawidłowe działanie operacji wymiany danych. W rzeczywistości może to być możliwe tylko dla stosunkowo niewielkiej liczby środowisk. W przypadku większego zbioru można ograniczyć testowanie do reprezentatywnej grupy wybranych środowisk. W ramach specyfikacji testów współdziałania należy zidentyfikować, skonfigurować i udostępnić zespołowi testowemu odpowiednie kombinacje środowisk docelowych. Następnie takie środowiska zostają przetestowane za pomocą wybranych funkcjonalnych przypadków testowych sprawdzających różnego rodzaju punkty wymiany danych w środowisku.

Współdziałanie wiąże się ze sposobem interakcji między różnymi modułami i systemami oprogramowania. Oprogramowanie o dobrych wskaźnikach współdziałania można zintegrować z wieloma innymi systemami bez wprowadzania w nim większych zmian i bez większego wpływu na zachowanie niefunkcjonalne. Jako miary współdziałania można użyć liczby koniecznych zmian i nakładu pracy potrzebnego na ich implementację i przetestowanie.

Testowanie współdziałania oprogramowania może na przykład koncentrować się na następujących cechach projektowych:

- wykorzystanie branżowych standardów komunikacji takich jak XML,
- zdolność automatycznego wykrywania potrzeb komunikacyjnych związanych z systemami, z którymi oprogramowanie współdziała, i odpowiedniego dostosowania mechanizmów komunikacji.

Testowanie współdziałania może być szczególnie istotne w przypadku następujących produktów:

- oprogramowania do powszechnej sprzedaży (ang. *Commercial of-the-shelf*) i gotowych narzędzi,
- aplikacji stanowiących systemy systemów,
- systemów wykorzystujących technologię Internetu rzeczy (IoT),
- usług internetowych komunikujących się z innymi systemami.

Tego rodzaju testy wykonuje się podczas testowania integracji modułów i testowania integracji systemów. Na poziomie testów integracyjnych systemów tego rodzaju testy przeprowadza się po ukończeniu systemu w celu ustalenia, jak dobrze współdziała on z innymi systemami. Systemy mogą współdziałać na różnych poziomach, więc analityk testów musi rozumieć te interakcje i umieć stworzyć warunki, które umożliwią przetestowanie różnych interakcji. Na przykład jeżeli między dwoma systemami ma zachodzić wymiana

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

danych, analityk testów musi być w stanie wygenerować niezbędne dane i transakcje potrzebne do dokonania takiej wymiany. Należy przy tym pamiętać, że nie wszystkie interakcje mogą być jasno zdefiniowane w dokumentacji wymagań. Definicje wielu z nich mogą się pojawić dopiero w dokumentacji architektury i projektu systemu. Analityk testów musi być przygotowany do przeanalizowania tych dokumentów w celu ustalenia punktów wymiany informacji między systemami oraz między systemem a jego środowiskiem, aby zapewnić przetestowanie wszystkich takich punktów. W testowaniu współdziałania stosuje się takie techniki jak podział na klasy równoważności, analiza wartości brzegowych, tablice decyzyjne, diagramy przejść pomiędzy stanami, przypadki użycia i testowanie sposobem par. Wykrywane defekty z reguły dotyczą niepoprawnej wymiany danych między modułami wchodzącymi ze sobą w interakcje.

4.2.5. Ocena użyteczności

Analityk testów często zajmuje się koordynowaniem i wsparciem procesu oceny użyteczności. Może to obejmować specyfikację testów użyteczności i występowanie w roli moderatora prowadzącego testy wspólnie z użytkownikami. Aby skutecznie realizować swoje zadania, analityk testów musi rozumieć główne aspekty, cele i podejścia związane z takim rodzajem testowania. Szczegółowe informacje wykraczające poza zakres niniejszej sekcji można znaleźć w specjalistycznym sylabusie ISTQB® dotyczącym testowania użyteczności [ISTQB_UT_SYL].

Istotne jest zrozumienie, dlaczego użytkownicy mają problemy z korzystaniem z systemu oraz dlaczego ich doświadczenia nie są pozytywne (np. podczas korzystania z oprogramowania do rozrywki). Aby to osiągnąć, należy przede wszystkim uwzględnić fakt, że pojęcie „użytkownik” może odnosić się do różnych profili osób (tzw. person), od ekspertów IT poprzez dzieci, aż po osoby niepełnosprawne.

4.2.5.1. Aspekty użyteczności

W tej sekcji zostaną omówione następujące aspekty użyteczności:

- użyteczność w rozumieniu standardu ISO 25010,
- doświadczenie użytkownika (UX) jako uogólnienie użyteczności,
- dostępność jako podcharakterystyka użyteczności.

Użyteczność

Testy użyteczności koncentrują się na defektach oprogramowania, które mają wpływ na możliwość wykonywania zadań przez użytkowników za pośrednictwem interfejsu użytkownika. Takie defekty mogą utrudnić użytkownikom realizację ich celów w sposób skuteczny, wydajny i satysfakcjonujący. Problemy dotyczące użyteczności mogą prowadzić do nieporozumień, błędów, opóźnień i zwykłych awarii w trakcie realizacji zadań przez użytkowników.

Poniżej podano listę podcharakterystyk użyteczności według standardu [ISO 25010] (ich definicje znajdują się w słowniku [ISTQB_GLOSSARY]):

- łatwość nauki,
- łatwość obsługi,
- estetyka interfejsu użytkownika (tj. atrakcyjność),
- ochrona przed błędami użytkownika,
- dostępność (patrz poniżej).

Doświadczenie użytkownika (UX)

Ocena doświadczenia użytkownika obejmuje całe doświadczenie użytkownika związane z przedmiotem testów, a nie tylko bezpośrednią interakcję. Jest to szczególnie istotne w przypadku przedmiotów testów, dla których kluczowym czynnikiem umożliwiającym sukces rynkowy jest przyjemność obsługi i satysfakcja użytkownika.

Typowe czynniki wpływające na doświadczenie użytkownika to m.in.:

- wizerunek marki (tj. użytkownicy mają zaufanie do producenta),

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

- interaktywne zachowanie,
- pomoc związana z przedmiotem testów (m.in. system pomocy, wsparcie i szkolenia).

Dostępność
Należy uwzględnić dostępność oprogramowania dla użytkowników, którzy mają szczególne potrzeby lub ograniczenia w korzystaniu z niego, m. in. osób niepełnosprawnych. Podczas testowania dostępności należy uwzględnić właściwe standardy, np. wytyczne Web Content Accessibility Guidelines (WCAG) oraz uregulowania prawne, takie jak ustawy o przeciwdziałaniu dyskryminacji ze względu na niepełnosprawność (Disability Discrimination Act — Irlandia Północna, Australia), ustawa o równości szans (Equality Act 2010 — Anglia, Szkocja, Walia) czy artykuł 508 (Section 508 — Stany Zjednoczone). Podobnie jak w przypadku użyteczności, dostępnością należy zainteresować się już w fazie projektowania. Testy zaczyna się często na poziomie testów integracyjnych i kontynuuje przez fazę testów systemowych do poziomu testów akceptacyjnych. O defektach mówi się z reguły wówczas, gdy oprogramowanie nie spełnia właściwych uregulowań prawnych lub standardów.

Typowe miary związane ze zwiększaniem dostępności koncentrują się na możliwości interakcji z aplikacją przez użytkowników niepełnosprawnych. Obejmuje to między innymi:
- funkcje rozpoznawania głosu do obsługi wprowadzania danych,
- zapewnienie prezentacji treści innych niż tekstowe w postaci odpowiednika tekstowego,
- możliwość zmiany rozmiaru tekstu bez utraty zawartości lub funkcjonalności.

Wytyczne dotyczące dostępności są pomocne dla analityka testów, ponieważ stanowią źródło informacji i list kontrolnych, które można wykorzystać podczas testowania (przykłady takich wytycznych można znaleźć w dokumencie [ISTQB_UT_SYL]). Ponadto dostępne są narzędzia i wtyczki do przeglądarek, które ułatwiają testerom wykrywanie problemów związanych z dostępnością, takich jak nieprawidłowy dobór palety kolorów na stronach internetowych, niezgodny z wytycznymi dotyczącymi dostępu dla osób dotkniętych ślepotą barw (daltonizmem).

4.2.5.2. Metody oceny użyteczności
Użyteczność, doświadczenie użytkownika i dostępność można testować, korzystając z jednej z następujących metod:
- testowania użyteczności,
- przeglądów użyteczności,
- ankiet i kwestionariuszy użytkowników.

Testowanie użyteczności
Podczas testowania użyteczności sprawdza się, czy użytkownicy mogą łatwo korzystać lub nauczyć się korzystać z systemu w celu osiągnięcia określonego celu w konkretnym kontekście. Testowanie użyteczności ma mierzyć następujące cechy:
- skuteczność — możliwość osiągnięcia przez użytkowników przy użyciu przedmiotu testów określonych celów z zachowaniem poprawności i kompletności w określonym kontekście użycia;
- efektywność — możliwość uzyskania przez użytkowników przy użyciu przedmiotu testów określonej efektywności w określonym kontekście użycia przy odpowiednich nakładach zużytych zasobów;
- satysfakcję — możliwość zadowolenia użytkowników z przedmiotu testów w określonym kontekście użycia.

Należy zauważyć, że projektowanie i specyfikowanie testów użyteczności analityk testów wykonuje często we współpracy z testerami dysponującymi specjalistycznymi kompetencjami w dziedzinie testowania użyteczności, a także z projektantami interfejsu obsługi użytkownika, którzy rozumieją proces projektowania zorientowany na użytkownika (więcej informacji zawiera dokument [ISTQB_UT_SYL]).

Przeglądy użyteczności
Inspekcje i przeglądy to forma testowania prowadzonego pod kątem oceny użyteczności, pozwalająca zwiększyć stopień zaangażowania użytkowników. To podejście może przyczynić się do obniżenia kosztów,

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

ponieważ umożliwia wykrycie problemów związanych z użytecznością w specyfikacji wymagań i w projektach na wczesnych etapach cyklu wytwarzania oprogramowania. W celu zidentyfikowania problemów z użytecznością podczas projektowania można wykorzystać ocenę heurystyczną (systematyczną inspekcję projektu interfejsu użytkownika pod kątem użyteczności), aby można było zaradzić tym problemom w ramach iteracyjnego procesu projektowania. Wymaga to zaangażowania niewielkiego zespołu oceniającego, który zbada interfejs i oceni jego zgodność z uznawanymi zasadami użyteczności („heurystyki"). Przeglądy zyskują na skuteczności wraz z widocznością interfejsu użytkownika. Na przykład, zwykle łatwiej zrozumieć i zinterpretować przykładowe zrzuty ekranów niż słowny opis funkcjonalności danego ekranu. Wizualizacja jest istotna dla dokonania odpowiedniego przeglądu dokumentacji pod kątem użyteczności.

Ankiety i kwestionariusze dotyczące użytkowników
Do gromadzenia obserwacji i informacji zwrotnych związanych z zachowaniem użytkowników systemu można zastosować techniki ankiet i kwestionariuszy. Standardowe, ogólnodostępne ankiety, takie jak inwentarz pomiarów użyteczności oprogramowania ang. *Software Usability Measurement Inventory* (SUMI) i inwentarz analizy i pomiaru stron internetowych ang. *Website Analysis and Measurement Inventory* (WAMMI) umożliwiają porównywanie rezultatów ze zgromadzonymi w bazie danych pomiarami dokonanymi w innych projektach. Co więcej, jako że ankieta SUMI uwzględnia konkretne metryki użyteczności, może zapewnić zestaw kryteriów zakończenia/akceptacji.

4.2.6. Testowanie przenaszalności

Testy przenaszalności dotyczą łatwości przenoszenia systemu lub modułu oprogramowania do docelowego środowiska, zarówno po raz pierwszy (jako nowa instalacja), jak i z istniejącego środowiska.

Klasyfikacja charakterystyk jakościowych produktów ISO 25010 obejmuje następujące podcharakterystyki przenaszalności:
- instalowalność,
- zdolność adaptacyjna,
- zastępowalność.

Zadania związane z identyfikacją ryzyka i projektowaniem testów charakterystyk przenaszalności realizują analityk testów i techniczny analityk testów (patrz [ISTQB_ALTTA_SYL], podrozdział 4.7.).

4.2.6.1. Testowanie instalowalności
Testowanie instalowalności przeprowadza się na oprogramowaniu z wykorzystaniem procedur używanych do jego instalowania i deinstalowania w docelowym środowisku.

Typowe cele testowania instalowalności, które powinien uwzględnić analityk testów, to między innymi:
- Sprawdzenie, czy różne konfiguracje oprogramowania można pomyślnie zainstalować. W sytuacjach, w których występuje wiele parametrów konfiguracyjnych, analityk testów może zastosować technikę testowania sposobem par w celu ograniczenia liczby kombinacji parametrów i skoncentrować się na szczególnie interesujących konfiguracjach (np. najczęściej używanych).
- Testowanie poprawności procedur instalacji i deinstalacji.
- Wykonywanie testów funkcjonalnych po instalacji lub deinstalacji w celu wykrycia ewentualnych, wprowadzonych defektów (np. niepoprawnych konfiguracji i niedostępnych funkcji).
- Identyfikowanie problemów związanych z użytecznością w procedurach instalacji i deinstalacji, na przykład w celu sprawdzenia, czy użytkownicy otrzymują podczas wykonywania procedury zrozumiałe instrukcje, odpowiedzi i komunikaty o błędach.

4.2.6.2. Testowanie zdolności adaptacyjnej
Testowanie zdolności adaptacyjnej umożliwia sprawdzenie, czy dana aplikacja może być zaadaptowana efektywnie i wydajnie do poprawnego funkcjonowania we wszystkich założonych środowiskach docelowych (sprzęt, oprogramowanie, warstwa pośrednia, system operacyjny, chmura itd.). Analityk testów wspiera testowanie zdolności adaptacyjnej poprzez identyfikację planowanych środowisk docelowych (np. wersje

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

obsługiwanych mobilnych systemów operacyjnych, różne wersje obsługiwanych przeglądarek) oraz poprzez projektowanie testów pokrywających kombinacje tych środowisk. Następnie środowiska docelowe zostają przetestowane za pomocą wybranych funkcjonalnych przypadków testowych sprawdzających różnego rodzaju komponenty obecne w środowisku.

4.2.6.3 Testowanie zastępowalności

Testowanie zastępowalności koncentruje się na możliwości zastąpienia w systemie modułów oprogramowania lub ich wersji przez inne moduły lub wersje. Może to być szczególnie przydatne w architekturach opartych na technologii Internetu rzeczy (IoT), w których często występuje wymiana różnych urządzeń i instalacja oprogramowania. Na przykład sprzęt używany w magazynie do rejestrowania i kontroli stanu zapasów może zostać zastąpiony przez bardziej zaawansowane urządzenie (np. zawierające lepszy skaner), a zainstalowane oprogramowanie może zostać zaktualizowane i zastąpione nową wersją, która umożliwia automatyczne tworzenie zamówień i wysyłanie ich do systemu dostawcy.
Testy zastępowalności mogą być wykonywane przez analityka testów równolegle z testami integracji funkcjonalności, jeśli do integracji z kompletnym systemem dostępny jest więcej niż jeden opcjonalny moduł.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

5. Przeglądy — 120 minut

Słowa kluczowe
przegląd oparty na liście kontrolnej

Cele nauczania dotyczące przeglądów

5.1. Wprowadzenie
Nie określono celów nauczania.

5.2 Korzystanie z list kontrolnych podczas przeglądów
TA-5.2.1. (K3) Kandydat potrafi zidentyfikować problemy w specyfikacji wymagań zgodnie z listą kontrolną podaną w sylabusie.
TA-5.2.2. (K3) Kandydat potrafi zidentyfikować problemy w historyjce użytkownika zgodnie z listą kontrolną podaną w sylabusie.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

5.1. Wprowadzenie

Analityk testów musi aktywnie uczestniczyć w procesie przeglądu i korzystać ze swojej unikalnej perspektywy. Prawidłowo dokonany przegląd może być najbardziej opłacalnym elementem, przyczyniającym się do ogólnej jakości dostarczanych produktów.

5.2. Korzystanie z list kontrolnych podczas przeglądów

Przeglądy oparte na listach kontrolnych to najczęściej stosowana przez analityków testów technika przeglądu podstawy testów. Podczas przeglądów używa się list kontrolnych, aby przypomnieć uczestnikom o sprawdzeniu w ramach przeglądu konkretnych punktów. Pomagają one również wyeliminować z przeglądu czynnik ludzki: „używamy tej samej listy we wszystkich przeglądach, nie tylko w odniesieniu do twojego produktu pracy".

Przegląd oparty na listach kontrolnych może być ogólny, do zastosowania we wszelkiego rodzaju przeglądach, lub skoncentrowany na określonych charakterystykach jakościowych, obszarach albo typach dokumentów. Na przykład ogólna lista kontrolna może służyć do weryfikacji ogólnych właściwości dokumentu, takich jak unikalny identyfikator, brak pozostawionych adnotacji „do uzupełnienia", właściwe formatowanie i tym podobne elementy zgodności z szablonem. Lista przeznaczona konkretnie do przeglądów dokumentacji wymagań może zawierać takie punkty, jak sprawdzenie odpowiedniego użycia terminów „będzie" i „powinien", testowalności każdego z uzgodnionych wymagań itp.

Również sam format wymagań może wskazywać, jakiego rodzaju lista kontrolna powinna zostać zastosowana. Do dokumentacji wymagań wyrażonych w języku naturalnym należy zastosować inne kryteria przeglądu niż do dokumentacji wymagań opartych na diagramach.

Listy kontrolne mogą być również ukierunkowane na określony aspekt, np.:

- kwalifikacje programisty/architekta albo kompetencje testera – w przypadku analityka testów odpowiednia będzie lista kontrolna zorientowana na kwalifikacje testerskie,
- określony poziom ryzyka (np. w systemach krytycznych ze względów bezpieczeństwa) — listy kontrolne zazwyczaj zawierają wówczas konkretne informacje związane z danym poziomem ryzyka,
- konkretną technikę testowania — lista kontrolna zawiera wówczas informacje niezbędne do zastosowania danej techniki (np. reguły, które mają być reprezentowane w tablicy decyzyjnej),
- określony element specyfikacji, np. wymaganie, przypadek użycia lub historyjka użytkownika — zagadnienia te omówiono w kolejnych podrozdziałach; takie listy kontrolne koncentrują się zwykle na innych elementach niż listy używane przez technicznego analityka testów do przeglądów kodu oraz architektury.

5.2.1. Przeglądy wymagań

Na liście kontrolnej do przeglądów wymagań mogą znaleźć się następujące elementy:

- źródło danego wymagania (np. osoba lub dział),
- testowalność każdego z wymagań,
- priorytet każdego wymagania,
- kryteria akceptacji dla każdego wymagania,
- dostępność struktury wywołań przypadku użycia, o ile ma ona zastosowanie,
- jednoznaczna identyfikacja każdego wymagania, przypadku użycia lub historyjki użytkownika,
- kontrola wersji każdego wymagania, przypadku użycia lub historyjki użytkownika,
- możliwość śledzenia każdego wymagania z wymagań biznesowych/marketingowych,
- możliwość śledzenia związków między wymaganiami a/lub przypadkami użycia (o ile są stosowane),
- zastosowanie spójnej terminologii (np. korzystanie ze słownika).

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

Należy pamiętać, że jeśli wymaganie nie jest testowalne — to znaczy jest zdefiniowane w taki sposób, że analityk testów nie może ustalić, w jaki sposób je przetestować — wówczas w tym wymaganiu występuje defekt. Na przykład wymaganie „Oprogramowanie powinno być bardzo przyjazne dla użytkownika" nie jest testowalne. Jak analityk testów ma ustalić, czy oprogramowanie jest przyjazne dla użytkownika, a tym bardziej „bardzo przyjazne dla użytkownika"? Gdyby zamiast tego wymaganie zawierało stwierdzenie „Oprogramowanie musi być zgodne ze standardami użyteczności określonymi w dokumencie standardów użyteczności, wersja xxx" i gdyby taki dokument standardów użyteczności istniał, wówczas to wymaganie byłoby testowalne. Jest to też wymaganie nadrzędne, ponieważ dotyczy wszystkich elementów interfejsu użytkownika. W rozbudowanej aplikacji mogłaby w takiej sytuacji istnieć konieczność wyprowadzenia z jednego wymagania wielu szczegółowych przypadków testowych. Kluczowe znaczenie miałoby również prześledzenie powiązań tego wymagania (lub standardów użyteczności z zewnętrznego dokumentu) z przypadkami testowymi, ponieważ w przypadku zmiany przywoływanej specyfikacji użyteczności należałoby dokonać przeglądu i odpowiednich modyfikacji wszystkich przypadków testowych.

Wymaganie jest również nietestowalne, jeżeli tester nie ma możliwości ustalenia, czy test został zaliczony, czy nie, lub nie jest w stanie zbudować testu, który może zostać zaliczony lub niezaliczony. Na przykład wymaganie „System będzie dostępny przez 100% czasu, 24 godziny na dobę, 7 dni w tygodniu, 365 (lub 366) dni w roku" jest nietestowalne.

Prosta lista kontrolna[2] do przeglądów przypadków użycia może zawierać następujące pytania:

- Czy podstawowe zachowanie (ścieżka) jest jasno zdefiniowane?
- Czy zostały zidentyfikowane wszystkie zachowania (ścieżki) alternatywne wraz z obsługą błędów?
- Czy zostały zdefiniowane komunikaty interfejsu użytkownika?
- Czy istnieje tylko jedna ścieżka podstawowa (tak powinno być – jeżeli jest inaczej, mamy do czynienia
  z więcej niż jednym przypadkiem użycia)?
- Czy każde z zachowań jest testowalne?

5.2.2. Przeglądy historyjek użytkownika

W projektach zwinnego wytwarzania oprogramowania (Agile) wymagania mają zwykle postać historyjek użytkownika. Takie historyjki reprezentują niewielkie, możliwe do zaprezentowania, wycinki funkcjonalności. Przypadek użycia opisuje transakcję użytkownika, obejmującą różne obszary funkcjonalne, natomiast historyjka użytkownika jest bardziej ograniczona, a jej zakres jest uzależniony od czasu potrzebnego na zaimplementowanie danej funkcjonalności. Lista kontrolna[3] do przeglądów historyjek użytkownika może zawierać następujące pytania:

- Czy historyjka jest odpowiednia dla docelowej iteracji/sprintu?
- Czy historyjka jest napisana z punktu widzenia osoby, która zgłosiła odpowiednie żądanie?
- Czy zostały zdefiniowane kryteria akceptacji i czy są one testowalne?
- Czy historyjka opisuje dobrze zdefiniowaną, odrębną funkcję?
- Czy dana historyjka jest niezależna od pozostałych?
- Czy historyjce przypisano priorytet?
- Czy historyjka została zapisana w powszechnie stosowanym formacie:
  „Jako <typ użytkownika> chcę <potrzeba>, żeby <cel do osiągnięcia>" [Cohn04]?

[2] W pytaniu egzaminacyjnym znajdzie się podzbiór elementów listy kontrolnej przypadków użycia, na podstawie którego należy udzielić odpowiedzi.
[3] W pytaniu egzaminacyjnym znajdzie się podzbiór elementów listy kontrolnej historyjek użytkownika, na podstawie którego należy udzielić odpowiedzi.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

Jeżeli w historyjce jest zdefiniowany nowy interfejs, wówczas wskazane jest zastosowanie ogólnej listy kontrolnej dla historyjek (takiej, jak podano powyżej) oraz szczegółowej listy kontrolnej dla interfejsów użytkownika.

5.2.3. Dostosowywanie list kontrolnych

Listę kontrolną można modyfikować (dostosowywać do potrzeb) w oparciu o następujące czynniki:

- organizacja (np. w celu uwzględnienia polityk, standardów i praktyk firmowych, ograniczeń prawnych),
- konkretny projekt lub konkretne prace rozwojowe (np. cel, standardy techniczne, czynniki ryzyka);
- produkt pracy poddawany przeglądowi (np. przeglądy kodu mogą być dostosowywane do specyfiki konkretnych języków programowania),
- poziom ryzyka dla produktu pracy poddawanego przeglądowi,
- techniki testowania, które mają zostać zastosowane.

Prawidłowo sporządzone listy kontrolne umożliwiają wykrycie problemów oraz ułatwiają rozpoczęcie dyskusji o dodatkowych elementach weryfikacji, które mogą nie być uwzględnione na liście. Łączenie różnych list kontrolnych jest dobrym sposobem na zapewnienie jak najwyższej jakości produktu pracy w wyniku przeglądu. Wykorzystanie standardowych list kontrolnych, takich jak listy przywoływane w sylabusie poziomu podstawowego, oraz opracowanie specyficznych dla danej organizacji list kontrolnych podobnych do tych wskazanych powyżej ułatwią analitykowi testów efektywne dokonywanie przeglądów.

Więcej informacji o przeglądach i inspekcjach można znaleźć w [Gilb93] i [Wiegers03]. Dalsze przykłady list kontrolnych znajdują się w źródłach przywoływanych w podrozdziale 7.4.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

6. Narzędzia testowe i automatyzacja testów — 90 minut

Słowa kluczowe

projekt testów, przygotowywanie danych testowych, skrypt testowy, testowanie oparte na słowach kluczowych, wykonywanie testu

Cele nauczania dotyczące narzędzi testowych i automatyzacji testów

6.1. Wprowadzenie

Nie określono celów nauczania.

6.2 Testowanie oparte na słowach kluczowych

TA-6.2.1. (K3) Dla danego scenariusza kandydat potrafi określić działania, które powinien podjąć analityk testów w projekcie testowym opartym na słowach kluczowych.

6.3 Rodzaje narzędzi testowych

TA-6.3.1. (K2) Kandydat potrafi wyjaśnić sposób wykorzystania i rodzaje narzędzi testowych stosowanych w projektowaniu testów, przygotowywaniu danych testowych i wykonywaniu testów.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

6.1. Wprowadzenie

Narzędzia testowe mogą znacznie zwiększyć efektywność i dokładność testowania. W niniejszym rozdziale opisano narzędzia testowe i podejścia do automatyzacji stosowane przez analityków testów. Warto podkreślić, że – tworząc rozwiązania dla testów automatycznych – analityk testów współpracuje z programistami, inżynierami testów automatycznych i technicznymi analitykami testów. Szczególnie znaczący jest udział analityka testów w testowaniu opartym na słowach kluczowych, w której wykorzystywana jest jego wiedza na temat dziedziny biznesowej i funkcjonalności systemu.

Dalsze informacje na temat automatyzacji testów i roli inżyniera testów automatycznych znajdują się w sylabusie ISTQB® poziomu zaawansowanego dla inżyniera testów automatycznych [ISTQB_TAE_SYL].

6.2. Testowanie oparte na słowach kluczowych

Testowanie oparte na słowach kluczowych jest jednym z podstawowych podejść stosowanych w automatyzacji testów. Do zadań analityka testów należy dostarczenie najważniejszych informacji wejściowych, czyli słów kluczowych i danych.

Słów kluczowych (czasem nazywanych słowami akcji) używa się przede wszystkim, choć nie wyłącznie, jako reprezentacji interakcji biznesowych z systemem na wysokim poziomie (np. „wycofaj zamówienie”). Każde słowo kluczowe reprezentuje zwykle szereg szczegółowych interakcji między aktorem a testowanym systemem. Sekwencje słów kluczowych (w tym odpowiednie dane testowe) służą do określania przypadków testowych [Buwalda02].

Podczas automatyzowania testu słowo kluczowe implementuje się w postaci jednego lub kilku wykonywalnych skryptów testowych. Narzędzia odczytują przypadki testowe zapisane jako ciąg słów kluczowych, które wywołują odpowiednie skrypty testowe implementujące funkcjonalność słowa kluczowego. Skrypty są zbudowane modularnie, aby łatwo je było odwzorowywać na konkretne słowa kluczowe. Do zaimplementowania takich modułowych skryptów konieczna jest umiejętność programowania.

Podstawowe korzyści płynące z testowania opartego na słowach kluczowych to:

- słowa kluczowe dotyczące konkretnej aplikacji lub dziedziny biznesowej mogą być definiowane przez ekspertów z danej dziedziny; specyfikowanie przypadków testowych może dzięki temu przebiegać efektywniej,
- osoba dysponująca głównie wiedzą dziedzinową może odnieść korzyść z automatycznie wykonywanych przypadków testowych (po zaimplementowaniu słów kluczowych w postaci skryptów) bez konieczności rozumienia kodu automatyzacji,
- modułowa technika pisania testów ułatwia efektywne utrzymanie przypadków testowych przez inżyniera testów automatycznych po wprowadzeniu zmian w funkcjonalności i interfejsie testowanego oprogramowania [Bath14],
- specyfikacja przypadków testowych jest niezależna od ich implementacji.

Do zadań analityka testów zwykle należy utworzenie i utrzymanie danych związanych ze słowami kluczowymi/słowami akcji. Należy pamiętać, że do zaimplementowania słów kluczowych niezbędne jest przygotowanie skryptów. Po zdefiniowaniu słów kluczowych i używanych danych osoba odpowiedzialna za automatyzację testów (np. techniczny analityk testów lub inżynier testów automatycznych) przekłada słowa kluczowe związane z procesem biznesowym i działania niższego poziomu na skrypty testów automatycznych.

Testowanie oparte na słowach kluczowych odbywa się z reguły w fazie testów systemowych, ale prace nad kodem mogą rozpocząć się już w fazie projektowania testów. W środowisku iteracyjnym, zwłaszcza

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

w wypadku stosowania mechanizmów ciągłej integracji i ciągłego wdrażania, projektowanie testów automatycznych jest również procesem ciągłym.

Po zdefiniowaniu wejściowych słów kluczowych i utworzeniu danych, analityk testów przejmuje odpowiedzialność za wykonanie skryptów testowych zawierających słowa kluczowe i za analizę wszelkich napotkanych awarii.

W przypadku wykrycia anomalii, analityk testów musi wziąć udział w badaniu przyczyny awarii, aby stwierdzić, czy defekt powodują słowa kluczowe, dane wejściowe, same skrypty testów automatycznych czy testowana aplikacja. Pierwszym krokiem analizy problemu jest z reguły wykonanie tego samego testu przy użyciu tych samych danych manualnie, aby sprawdzić, czy awaria jest spowodowana przez aplikację. Jeżeli tym razem awaria nie wystąpi, analityk testów powinien przeanalizować sekwencję testów, która doprowadziła do awarii, aby sprawdzić, czy problem nie wystąpił w jednym z wcześniejszych kroków (np. zostały wprowadzone niepoprawne dane wejściowe), a defekt ujawnił się później w toku przetwarzania. Jeżeli analityk testów nie jest w stanie ustalić przyczyny awarii, to zgromadzone podczas analizy problemu informacje powinny zostać przekazane technicznemu analitykowi testów lub programiście do dalszego zbadania.

6.3. Rodzaje narzędzi testowych

Znaczna część prac wykonywanych przez analityka testów wymaga skutecznego wykorzystania narzędzi. Tę skuteczność potęgują jego:

- wiedza, jakie narzędzia należy zastosować,
- świadomość, że dzięki zastosowaniu narzędzi można zwiększyć efektywność wysiłku testowego (np. by zapewnić lepsze pokrycie w przewidzianym czasie).

6.3.1. Narzędzia do projektowania testów

Narzędzia do projektowania testów ułatwiają tworzenie przypadków testowych i danych testowych używanych podczas testowania. Narzędzia te mogą korzystać z dokumentacji wymagań w określonych formatach, modeli (np. UML) lub z danych wprowadzanych przez analityka testów. Narzędzia do projektowania testów są często projektowane i budowane z myślą o współdziałaniu z konkretnymi formatami i określonymi narzędziami takimi jak konkretne narzędzia do zarządzania wymaganiami.

Narzędzia do projektowania testów mogą dostarczyć analitykowi testów informacje przydatne do ustalenia typów testów, które są konieczne do osiągnięcia żądanego poziomu pokrycia, poziomu zaufania do systemu lub wykonania działań związanych z łagodzeniem ryzyka. Na przykład narzędzia drzewa klasyfikacji generują (i wyświetlają) zestawy kombinacji potrzebnych do uzyskania pełnego pokrycia zgodnie z wybranym kryterium pokrycia. Na podstawie tych informacji analityk testów może następnie ustalić, jakie przypadki testowe muszą zostać wykonane.

6.3.2. Narzędzia do przygotowywania danych testowych

Narzędzia do przygotowywania danych testowych przynoszą następujące korzyści:

- Możliwość przeanalizowania dokumentów takich jak dokument wymagań, a nawet kodu źródłowego, w celu ustalenia danych wymaganych do osiągnięcia wymaganego poziomu pokrycia produktu podczas testowania.
- Możliwość pobrania zestawu danych z systemu produkcyjnego i „wyczyszczenia” lub anonimizacji w celu usunięcia wszelkich danych osobowych, jednak z zachowaniem wewnętrznej spójności danych. Wyczyszczone dane można następnie wykorzystać do testowania, nie ryzykując wycieku lub niezgodnego z przeznaczeniem wykorzystania danych osobowych. Jest to szczególnie ważne w sytuacjach, gdy potrzebne są duże ilości realistycznych danych i gdy istnieje ryzyko związane z zabezpieczeniami i prywatnością danych.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

- Możliwość wygenerowania syntetycznych danych testowych na podstawie podanych zestawów parametrów wejściowych (np. do testów losowych). Niektóre takie narzędzia umożliwiają przeanalizowanie struktury bazy danych w celu ustalenia, jakie dane wejściowe powinien podać analityk testów.

6.3.3 Narzędzia do wykonywania testów automatycznych

Narzędzia do wykonywania testów są używane na wszystkich poziomach testów w celu wykonania testów automatycznych i sprawdzenia ich rzeczywistych rezultatów. Narzędzia do wykonywania testów wykorzystuje się z reguły do jednego lub więcej z następujących celów:

- obniżenia kosztów (nakładu pracy i/lub czasu),
- wykonania większej liczby testów,
- wykonania tego samego testu w różnych środowiskach,
- zapewnienia większej powtarzalności wykonywania testów,
- wykonania testów, których nie da się wykonać manualnie (np. testów walidacji dużych zbiorów danych).

Cele te często nakładają się na główne cele zwiększenia pokrycia przy jednoczesnym obniżeniu kosztów.

Najwyższy zwrot z inwestycji w narzędzia do automatyzacji testów uzyskuje się z reguły w przypadku automatyzowania testów regresji ze względu na niewielkie zapotrzebowanie na ich pielęgnację oraz na ich powtarzalne wykonywanie. Efektywna jest też automatyzacja testów dymnych ze względu na ich częste wykonywanie, potrzebę szybkiego uzyskania rezultatów testu oraz, mimo potencjalnie wyższych kosztów pielęgnacji, możliwość automatycznego oceniania nowych wersji oprogramowania w środowisku ciągłej integracji.

Narzędzia do wykonywania testów powszechnie wykorzystuje się w testach systemowych i integracyjnych. Niektóre narzędzia, zwłaszcza narzędzia do testów interfejsów API, można stosować również w testach modułowych. Wykorzystanie narzędzi tam, gdzie są one najbardziej przydatne, pomoże zwiększyć zwrot z inwestycji.

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

7. Dokumenty pomocnicze

7.1. Standardy

- [ISO25010] ISO/IEC 25010 (2011) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models (Rozdział 4.)
- [ISO29119-4] ISO/IEC/IEEE 29119-4 Software and Systems Engineering – Software Testing — Part 4, Test Techniques, 2015
- [OMG-DMN] Object Management Group: OMG® Decision Model and Notation™, Version 1.3., December 2019; url: www.omg.org/spec/DMN/ (Rozdział 8.)
- [OMG-UML] Object Management Group: OMG® Unified Modeling Language®, Version 2.5.1., December 2017; url: www.omg.org/spec/UML/
- [RTCA DO-178C/ED-12C] Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12C, 2013 (Rozdział 1.)

7.2. Dokumenty ISTQB® i IREB®

- [IREB_CPRE] IREB® Certified Professional for Requirements Engineering sylabus poziomu podstawowego, wersja 2.2.2., 2017
- [ISTQB_AL_OVIEW] Wprowadzenie do poziomu zaawansowanego ISTQB®, wersja 2.0
- [ISTQB_ALTTA_SYL] Sylabus ISTQB® poziomu zaawansowanego — Techniczny analityk testów, wersja 2019
- [ISTQB_FL_SYL] Sylabus ISTQB® poziomu podstawowego, wersja 2018
- [ISTQB_GLOSSARY] Słownik wyrażeń związanych z testowaniem, https://glossary.istqb.org/
- [ISTQB_TAE_SYL] ISTQB® Advanced Level Test Automation Engineer Syllabus, v. 2017
- [ISTQB_UT_SYL] ISTQB® Foundation Level Specialist Syllabus Usability Testing, v. 2018

7.3. Książki i artykuły

- [Bath14] Graham Bath, Judy McKay, „The Software Test Engineer's Handbook (2nd Edition)", Rocky Nook, 2014, ISBN 978-1-933952-24-6
- [Beizer95] Boris Beizer, „Black-box Testing", John Wiley & Sons, 1995, ISBN 0-471-12094-4
- [Black02] Rex Black, „Managing the Testing Process (2nd Edition)", John Wiley & Sons: New York, 2002, ISBN 0-471-22398-0
- [Black07] Rex Black, „Pragmatic software testing: Becoming an effective and efficient test professional", John Wiley and Sons, 2007, ISBN 978-0-470-12790-2
- [Black09] Rex Black, „Advanced Software Testing, Volume 1", Rocky Nook, 2009, ISBN 978-1-933-952-19-2
- [Buwalda02] Hans Buwalda, „Integrated Test Design and Automation: Using the Test Frame Method", Addison-Wesley Longman, 2002, ISBN 0-201-73725-6
- [Chow1978] T.S. Chow, Testing Software Design Modeled by Finite-State Machines, IEEE Transactions on Software Engineering vol. SE-4, issue 3, May 1978, pp. 178-187
- [Cohn04] Mike Cohn, „User Stories Applied: For Agile Software Development", Addison-Wesley Professional, 2004, ISBN 0-321-20568-5
- [Copeland04] Lee Copeland, „A Practitioner's Guide to Software Test Design", Artech House, 2004, ISBN 1-58053-791-X
- [Craig02] Rick David Craig, Stefan P. Jaskiel, „Systematic Software Testing", Artech House, 2002, ISBN 1-580-53508-9
- [Forgács19] István Forgács, Attila Kovács, "Practical Test Design", BCS, 2019, ISBN 978-1-780-1747-23

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

- [Gilb93] Tom Gilb, Dorothy Graham, „Software Inspection", Addison-Wesley, 1993, ISBN 0-201-63181-4
- [Koomen06] Tim Koomen, Leo van der Aalst, Bart Broekman, Michiel Vroon „TMap NEXT, for result driven testing", UTN Publishers, 2006, ISBN 90-72194-80-2
- [Kuhn16] Richard Kuhn et al, „Introduction to Combinatorial Testing", CRC Press, 2016, ISBN 978-0-429-18515-1
- [Mosley93] Daniel J. Mosley, The Handbook of MIS Application Software Testing, Yourdon Press, Prentice-Hall. 1993, ISBN 978-0-13-907007-5
- [Myers11] Glenford J. Myers, „The Art of Software Testing" 3rd Edition, John Wiley & Sons, 2011, ISBN: 978-1-118-03196-4
- [Offutt16] Jeff Offutt, Paul Ammann, „Introduction to Software Testing" 2nd Edition, Cambridge University Press, 2016, ISBN 13:978-1-107-17201-2,
- [Roman18] Adam Roman, „Testowanie i jakość oprogramowania. Modele, techniki, narzędzia", PWN, 2018, ISBN 978-83-01-19644-8
- [vanVeenendaal12] Erik van Veenendaal, „Practical risk-based testing. Product Risk Management: The PRISMA Method", UTN Publishers, 2012, ISBN 978-94-9098-607-0
- [Wiegers03] Karl Wiegers, „Software Requirements", Microsoft Press, 2003, ISBN 0-735-61879-8
- [Whittaker03] James Whittaker, „How to Break Software", Addison-Wesley, 2003, ISBN 0-201-79619-8
- [Whittaker09] James Whittaker, „Exploratory software testing: tips, tricks, tours, and techniques to guide test design", Addison-Wesley, 2009, ISBN 0-321-63641-4

7.4. Inne dokumenty pomocnicze

Następujące odwołania wskazują informacje dostępne w Internecie i w innych źródłach. Odwołania zostały sprawdzone w momencie publikacji niniejszego sylabusa poziomu zaawansowanego, ISTQB® nie ponosi jednak odpowiedzialności za ich ewentualną późniejszą niedostępność.

- (Rozdział 3.)
  – Czerwonka, Jacek: www.pairwise.org
  – Przykładowa taksonomia defektów oparta na pracach Borisa Beizera: inet.uni2.dk/~vinter/bugtaxst.doc
  – Dobry przegląd różnych taksonomii: testingeducation.org/a/bugtax.pdf
  – Bach, James: Heuristic Risk-Based Testing
  – Exploring Exploratory Testing, Cem Kaner i Andy Tinkham, www.kaner.com/pdfs/ExploringExploratoryTesting.pdf
  – Pettichord, Bret, „An Exploratory Testing Workshop Report", www.testingcraft.com/exploratorypettichord
- (Rozdział 5.)
  – http://www.tmap.net/checklists-and-templates

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

8. Załącznik A

Poniższa tabela została opracowana na podstawie pełnej tabeli zawartej w standardzie ISO 25010. W tabeli skupiono się jedynie na charakterystykach jakościowych omówionych w sylabusie dla analityka testów. Zestawiono w niej terminy pochodzące ze standardu ISO 9126 (używane w wersji 2012 sylabusa) z terminami pochodzącymi z nowszego standardu ISO 25010 (używanymi w niniejszej wersji sylabusa).

ISO/IEC 25010	ISO/IEC 9126-1	Uwagi
Funkcjonalność (przydatność funkcjonalna)	Funkcjonalność	
Kompletność funkcjonalna		
Poprawność funkcjonalna	Dokładność	
Adekwatność funkcjonalna	Dopasowanie	
	Współdziałanie	Charakterystyka przeniesiona do kompatybilności
Użyteczność		
Stosowność	Zrozumiałość	Nowa nazwa
Łatwość nauki	Łatwość nauki	
Łatwość obsługi	Łatwość obsługi	
Ochrona przed błędami użytkownika		Nowa charakterystyka podrzędna
Estetyka interfejsu użytkownika	Atrakcyjność	Nowa nazwa
Dostępność		Nowa charakterystyka podrzędna
Kompatybilność		Nowa definicja
Współdziałanie		
Współistnienie		Charakterystyka omawiana w sylabusie dla technicznego analityka testów

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

9. Indeks

0-przełączenie, 33, 34
adekwatność funkcjonalna, 44, 45, 62
analiza ryzyka, 20
analiza testów, 14, 15, 16, 19, 20
analiza wartości brzegowych, 29, 30, 33, 48
anonimizacja, 58
ataki usterek, 22
atrakcyjność, 62
autentykacja, 45
bezpieczeństwo, 24, 25
charakterystyki jakościowe, 19, 28, 45, 46, 53, 62
  testowanie, 46
cykl wytwarzania oprogramowania, 14, 15, 19, 24, 45, 46, 47, 50
czarnoskrzynkowa technika testowania, 28, 35, 40, 42, 43
dane testowe, 17, 21, 26, 28, 35, 57, 58, 59
diagram przepływu sterowania, 38
diagram stanów, 33, 34
dojrzałość, 45
dokładność, 30, 46, 62
  testowania, 38, 57
dopasowanie, 62
dostępność, 45, 48, 49
doświadczenie użytkownika, 48
drzewo klasyfikacji, 35, 36, 37
estetyka interfejsu użytkownika, 45, 48, 62
funkcjonalność, 45, 62
harmonogram wykonywania testów, 20, 21, 22
heurystyka, 50
historyjka użytkownika, 16, 28, 40, 41, 46, 47, 53, 54, 55
  przegląd, 54
implementacja testów, 14, 15, 20, 26, 32
instalowalność, 45, 50
integralność, 45
inwentarz analizy i pomiaru stron internetowych (WAMMI), 50
inwentarz analizy i pomiaru użyteczności (SUMI), 50
karta opisu testu, 21, 39, 41, 42
klasa równoważności, 30, 35, 36, 38
kompletność funkcjonalna, 45, 62
kryterium wyjścia, 15, 20, 21
lista kontrolna, 40, 49
  dostosowanie, 55
łatwość nauki, 45, 48, 62
łatwość obsługi, 45, 48, 62
łatwość ponownego użycia, 45
łączenie list kontrolnych, 55
łączenie technik, 38
model cech, 36
modułowość, 45
modyfikowalność, 45
monitorowanie i nadzór nad testami, 14
narzędzia, 21, 34, 36, 37, 41, 42, 45, 49, 57
  do projektowania testów, 58
  do przygotowywania danych testowych, 58
  do wykonywania testów automatycznych, 59
  do zarządzania testami, 47
niezaprzeczalność, 45
niezawodność, 45
N-przełączenie, 33, 34
ocena heurystyczna, 50
ocena użyteczności, 48
ochrona przed błędami użytkownika, 45, 48, 62
odtwarzalność, 45
osiągalność, 45
pielęgnowalność, 45
plan testów, 16, 20, 25
planowanie testów, 14
podcharakterystyki jakościowe, 45
podlegający analizie, 45
podstawa testów, 16, 17, 18, 19, 20, 22, 42
  funkcjonalności, 46
  przegląd, 53
  wydajnościowych, 37
podział na klasy równoważności, 28, 33, 48
pokrycie N-przełączeń, 34
pokrycie okrążenia, 34
poprawność funkcjonalna, 24, 45, 46, 62
poufność, 45
procedura testowa, 20, 22, 28, 40
projektowanie przypadków testowych, 19, 34
projektowanie testów, 14, 15, 16, 17, 28, 57, 58
przegląd, 53
przegląd historyjek użytkownika, 54
przegląd oparty na liście kontrolnej, 53
przegląd wymagań, 53
przenaszalność, 24, 45, 46, 50
przepustowość, 45
przydatność funkcjonalna, 62
przypadek testowy, 17, 20, 30, 38, 40, 42

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

łagodzenie ryzyka, 25
niskiego poziomu, 18
podstawa, 16
priorytetyzacja, 20
projektowanie, 17, 19
wysokiego poziomu, 18
zmniejszanie liczby, 28, 32
przypadek użycia, 53
rozliczalność, 45
ryzyko, 24
analiza, 39
identyfikacja, 24
łagodzenie, 25, 58
ocena, 25, 26
podejście w głąb, 26
podejście wszerz, 26
poziom, 25, 26, 53
prawdopodobieństwo, 25
produktowe, 16
projektowe, 19
wpływ, 25
skrypt testowy, 17, 20, 57
słowa akcji, *Patrz* słowa kluczowe
słowa kluczowe, 21, 57, 58
standard
DO-178C, 21
ED 12C, 21
ISO 25010, 19, 45, 48, 50
ISO 29119-4, 28
ISO 9126, 62
OMG DMN, 31
OMG UML, 37
startegia testów, 16
sterownik, 19
stosowność, 45, 62
strategia testów, 14, 25, 45
oparta na ryzyku, 21, 24
reaktywna, 21
śledzenie, 17, 18, 19, 47, 53
środowisko docelowe, 47, 50
środowisko testowe, 14, 20, 21
lokalne, 19
projektowanie, 17
tablica decyzyjna, 31, 38, 48
definiowanie, 32
minimalizowanie, 32
z ograniczonym zakresem wejść, 31
z rozszerzonym zakresem wejść, 31
tablica stanów, 33, 34
taksonomia defektów, 40, 42, 61
technika testowania, 27
techniki kombinatoryczne, 29
testowalność, 45
testowanie adekwatności funkcjonalnej, 46
testowanie eksploracyjne, 22, 41, 42
testowanie kompletności funkcjonalnej, 47
testowanie oparte na defektach, 42
testowanie oparte na doświadczeniu, 38, 43
testowanie oparte na przypadkach użycia, 37, 48
testowanie oparte na ryzyku, 24, 26
testowanie oparte na słowach kluczowych, 57
testowanie poprawności funkcjonalnej, 46
testowanie przejść pomiędzy stanami, 33, 48
testowanie przenaszalności, 50
testowanie sposobem par, 29, 35, 36, 48, 50
testowanie użyteczności, 45
testowanie w cyklu wytwarzania oprogramowania, 14
testowanie w oparciu o listę kontrolną, 40
testowanie w oparciu o tablicę decyzyjną, 31
testowanie współdziałania, 47
testowanie zastępowalności, 51
testowanie zdolności adaptacyjnej, 50
testy swobodne, 21
tolerowanie usterek, 45
ukończenie testów, 14
ułatwienia dostępu, 62
UX (User Experience), 48
użyteczność, 24, 45, 48, 62
ankiety i kwestionariusze, 50
ocena, 48
przeglądy, 49
testowanie, 49
warunek dozoru, 33, 34
warunek testowy, 16, 17, 18, 19, 20, 21, 28, 32, 42
warunek wstępny, 19, 20, 35
współdziałanie, 45, 62
współistnienie, 45, 62
wybór najlepszej techniki, 38
wydajność, 45
wykonanie testów, 15
wykonywanie testów, 14, 22
wykorzystanie zasobów, 45
wymagania, 16, 19, 25, 28, 32, 46, 47
brakujące, 20
przegląd, 53
testowalne, 53
wymaganie, 53
wyrocznia testowa, 19, 46
zabezpieczenia, 42, 45, 58
zachowanie w czasie, 45
zastępowalność, 45, 50, 51
zaślepka, 19
zdolność adaptacyjna, 45, 50
zestaw testowy, 20, 28, 38, 43
zgadywanie błędów, 22, 39

© International Software Testing Qualifications Board Stowarzyszenie Jakości Systemów Informatycznych

zgodność, 45, 62
zrozumiałość, 62
zwinne wytwarzanie oprogramowania, 15, 16, 19, 24, 25, 41, 46, 54

© International Software Testing Qualifications Board
Stowarzyszenie Jakości Systemów Informatycznych