Certyfikowany tester

Sylabus dla poziomu zaawansowanego
Analityk Testów

Wersja 2012

International Software Testing Qualifications Board®
© Stowarzyszenie Jakości Systemów Informatycznych

Prawa autorskie
Niniejszy dokument może być kopiowany w całości lub publikowany w wybranych fragmentach z podaniem źródła.

Informacje o prawach autorskich
Kopiowanie niniejszego dokumentu w całości lub w wybranych fragmentach jest dozwolone, o ile zostanie wskazane źródło.

Copyright © International Software Testing Qualifications Board (w dalszej części dokumentu nazywana ISTQB®).

Podgrupa robocza poziomu zaawansowanego dla Analityka Testów: Judy McKay (przewodnicząca), Mike Smith, Erik Van Veenendaal; 2010-2012.

Tłumaczenie: BTInfo Biuro Tłumaczeń Informatycznych.

Przegląd: Michał Dudziak, Krystian Kaczor, Mirosław Panek, Jakub Rosiński, Jan Sabak, Radosław Smilgin.

Historia zmian

Wersja	Data	Uwagi
GA 2012	24.02.2021	Przegląd edytorski
GA 2012	19.10.2012	Ostateczne poprawki redakcyjne i oczyszczenie dokumentu przed wydaniem firmowanym przez zgromadzenie ogólne (GA)
RC 2012	15.08.2012	Wersja kandydująca z uwzględnionymi ostatecznymi modyfikacjami zgłoszonymi przez rady krajowe
Beta 2012	27.06.2012	Uwzględnienie komentarzy do glosariusza i EWG
Beta 2012	08.06.2012	Zredagowana wersja udostępniona radom krajowym
Beta 2012	07.04.2012	Wersja beta przekazana zgromadzeniu ogólnemu
Beta 2012	07.04.2012	Uwzględnienie komentarzy rad krajowych zgłoszonych do wydania alfa
Alfa 2012	09.03.2012	Uwzględnienie komentarzy rad krajowych zgłoszonych do październikowego wydania
D2011	23.10.2011	Rozdzielenie sylabusów, modyfikacje celów nauczania i dostosowanie tekstu do celów nauczania. Dodanie celów biznesowych.
D101227	27.12.2010	Zaakceptowanie zmian formatowania i poprawek niemających wpływu na znaczenie zdań
D100626	26.06.2010	Uwzględnienie zmian zaakceptowanych w 2009 r., oddzielenie rozdziałów dotyczących poszczególnych modułów
V2007	12.10.2007	Sylabus certyfikowanego testera dla poziomu zaawansowanego, wersja 2007
ISTQB 1.2E	09.2003	Sylabus ISTQB dla poziomu zaawansowanego EOQ-SG
ISEB 1.1	04.09.2001	Sylabus ISEB Practitioner

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Spis treści

Historia zmian 3
Spis treści 4
Podziękowania 7
0. Wprowadzenie 8
0.1 Przeznaczenie dokumentu 8
0.2 Opis 8
0.3 Cele nauczania podlegające egzaminowaniu 8
1. Proces testowy — 300 minut 9
1.1 Wprowadzenie 10
1.2 Testowanie w cyklu życia oprogramowania 10
1.3 Planowanie, monitorowanie i nadzór nad testami 12
1.3.1 Planowanie testów 12
1.3.2 Monitorowanie i nadzór nad testami 12
1.4 Analiza testów 13
1.5 Projektowanie testów 14
1.5.1 Przypadki testowe konkretne i logiczne 14
1.5.2 Tworzenie przypadków testowych 15
1.6 Implementacja testów 16
1.7 Wykonanie testów 18
1.8 Ocena kryteriów wyjścia i raportowanie 19
1.9 Czynności związane z zakończeniem testów 20
2. Zarządzanie testami: zadania Analityka Testów — 90 minut 21
2.1 Wprowadzenie 22
2.2 Monitorowanie i nadzór nad postępem testów 22
2.3 Testowanie rozproszone, zlecone na zewnątrz, zlecone wewnątrz 23
2.4 Zadania Analityka Testów w testowaniu opartym na ryzyku 23
2.4.1 Wprowadzenie 23
2.4.2 Identyfikacja ryzyka 24
2.4.3 Ocena ryzyka 24
2.4.4 Łagodzenie ryzyka 25
3. Techniki testowania — 825 minut 27
3.1 Wprowadzenie 29
3.2 Techniki oparte na specyfikacji 29
3.2.1 Klasy równoważności 29
3.2.2 Analiza wartości brzegowych 30

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

3.2.3 Tablice decyzyjne ... 31
3.2.4 Tworzenie grafów przyczynowo-skutkowych ... 32
3.2.5 Testowanie przejść pomiędzy stanami ... 32
3.2.6 Techniki testowania kombinatoryjnego ... 34
3.2.7 Testowanie w oparciu o przypadki użycia ... 35
3.2.8 Testowanie w oparciu o historyjki użytkownika ... 35
3.2.9 Analiza dziedzinowa ... 36
3.2.10 Łączenie technik ... 37
3.3 Techniki oparte na defektach ... 37
3.3.1 Korzystanie z technik opartych na defektach ... 37
3.3.2 Taksonomie defektów ... 38
3.4 Techniki oparte na doświadczeniu ... 39
3.4.1 Zgadywanie błędów ... 39
3.4.2 Testowanie w oparciu o listy kontrolne ... 40
3.4.3 Testowanie eksploracyjne ... 41
3.4.4 Wybór najlepszej techniki ... 42
4. Testowanie atrybutów jakościowych oprogramowania — 120 minut ... 43
4.1 Wprowadzenie ... 44
4.2 Charakterystyki jakościowe w testowaniu dziedziny biznesowej ... 45
4.2.1 Testowanie dokładności ... 46
4.2.2 Testowanie dopasowania ... 46
4.2.3 Testowanie współdziałania ... 46
4.2.4 Testowanie użyteczności ... 47
4.2.5 Testowanie dostępności ... 49
5. Przeglądy — 165 minut ... 50
5.1 Wprowadzenie ... 51
5.2 Korzystanie z list kontrolnych podczas przeglądów ... 51
6. Zarządzanie defektami — 120 minut ... 54
6.1 Wprowadzenie ... 55
6.2 Kiedy można wykryć defekt? ... 55
6.3 Pola zgłoszenia defektu ... 55
6.4 Klasyfikacja defektów ... 56
6.5 Analiza przyczyny podstawowej ... 57
7. Narzędzia testowe — 45 minut ... 59
7.1 Wprowadzenie ... 60
7.2 Narzędzia testowe i automatyzacja testów ... 60
7.2.1 Narzędzia do projektowania testów ... 60
7.2.2 Narzędzia do przygotowywania danych testowych ... 60

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

7.2.3 Narzędzia do automatyzacji testów 60
8. Dokumenty pomocnicze 65
8.1 Normy 65
8.2 Dokumenty ISTQB 65
8.3 Książki 65
8.4 Inne dokumenty pomocnicze 66
9. Indeks 67

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Podziękowania

Niniejszy dokument został opracowany przez zespół członków podgrupy roboczej International Software Testing Qualifications Board dla poziomu zaawansowanego dla modułu Analityk Testów: Judy McKay (przewodnicząca), Mike'a Smitha, Erika Van Veenendaala.

Zespół składa podziękowania dla zespołu recenzentów oraz komitetom krajowym za sugestie i wskazówki.

W chwili zakończenia prac nad sylabusem poziomu zaawansowanego członkami grupy roboczej poziomu zaawansowanego byli (w kolejności alfabetycznej):

Graham Bath, Rex Black, Maria Clara Choucair, Debra Friedenberg, Bernard Homès (wiceprzewodniczący), 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.

Następujące osoby uczestniczyły w przeglądach, zgłaszały komentarze i głosowały nad niniejszym sylabusem:

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

Niniejszy dokument został formalnie wydany przez zgromadzenie ogólne ISTQB® 19 października 2012r.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

0. Wprowadzenie

0.1 Przeznaczenie dokumentu

Niniejszy sylabus stanowi podstawę egzaminu International Software Testing Qualification dla Analityków Testów na poziomie zaawansowanym. ISTQB® udostępnia ten sylabus następującym odbiorcom:

1. Radom krajowym (National Boards) w celu tłumaczenia na języki lokalne i akredytacji dostawców szkoleń. Rady krajowe mogą dostosowywać sylabus do potrzeb danego języka i modyfikować odwołania do literatury tak, aby wskazywały na publikacje lokalne.
2. Komisjom egzaminacyjnym (Exam Boards), jako podstawę do formułowania pytań egzaminacyjnych w języku lokalnym, odpowiadających celom nauczania danego sylabusa.
3. Dostawcom szkoleń w celu tworzenia materiałów szkoleniowych i doboru odpowiednich metod nauczania.
4. Kandydatom ubiegającym się o certyfikat w celu przygotowania do egzaminu (w ramach szkoleń zorganizowanych lub samodzielnie).
5. Międzynarodowej społeczności zajmującej się inżynierią oprogramowania i inżynierią systemów w celu rozwijania zawodu testera oprogramowania i systemów oraz jako podstawę książek i artykułów.

ISTQB® może zezwolić innym podmiotom na korzystanie z niniejszego sylabusa do innych celów, pod warunkiem uzyskania wcześniejszej pisemnej zgody.

0.2 Opis

Poziom zaawansowany jest podzielony na trzy osobne sylabusy:

- Kierownik Testów
- Analityk Testów
- Techniczny Analityk Testów

Dokument Advanced Level Overview [ISTQB_AL_OVIEW] zawiera następujące informacje:

- cele biznesowe dla każdego sylabusa,
- podsumowanie każdego sylabusa,
- powiązania między sylabusami,
- opis poziomów wiedzy (poziomy K),
- załączniki.

0.3 Cele nauczania podlegające egzaminowaniu

Cele nauczania wspierają cele biznesowe i służą do formułowania egzaminów certyfikacyjnych „Analityk Testów — poziom zaawansowany”. Co do zasady, wszystkie części niniejszego sylabusa podlegają weryfikacji na poziomie K1, czyli kandydat musi potrafić rozpoznać i pamiętać pojęcie lub koncepcję. Cele nauczania na poziomach K2, K3 i K4 są przedstawione na początku każdego z rozdziałów.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

1. Proces testowy — 300 minut

Słowa kluczowe

konkretny przypadek testowy, kryteria wyjścia, przypadek testowy wysokiego poziomu, logiczny przypadek testowy, przypadek testowy niskiego poziomu, nadzór nad testami, projekt testu, wykonanie testów, implementacja testów, planowanie testów

Cele nauczania dotyczące procesu testowego

1.2 Testowanie w cyklu życia oprogramowania

TA-1.2.1 (K2) Kandydat potrafi wyjaśnić, jak i dlaczego moment i zakres zaangażowania Analityka Testów różnią się w zależności od przyjętego modelu cyklu życia oprogramowania

1.3 Planowanie, monitorowanie i nadzór nad testami

TA-1.3.1 (K2) Kandydat potrafi podsumować czynności wykonywane przez Analityka Testów związane z zaplanowaniem i zapewnieniem nadzoru nad testami

1.4 Analiza testów

TA-1.4.1 (K4) Kandydat potrafi dokonać analizy przedstawionego scenariusza, w tym opisu projektu i modelu cyklu życia oprogramowania, w celu ustalenia odpowiednich zadań realizowanych przez Analityka Testów w fazach analizy i projektowania

1.5 Projektowanie testów

TA-1.5.1 (K2) Kandydat potrafi wyjaśnić, dlaczego warunki testowe powinny być zrozumiałe dla interesariuszy

TA-1.5.2 (K4) Kandydat potrafi dokonać analizy scenariusza projektowego w celu ustalenia optymalnego zastosowania przypadków testowych niskiego poziomu (konkretnych) i wysokiego poziomu (logicznych)

1.6 Implementacja testów

TA-1.6.1 (K2) Kandydat potrafi opisać typowe kryteria wyjścia dla analizy testów i projektowania testów oraz wyjaśnić, w jaki sposób spełnienie tych kryteriów wpływa na nakład pracy przy implementacji testów

1.7 Wykonanie testów

TA-1.7.1 (K3) Dla podanego scenariusza kandydat potrafi określić kroki wykonania testów oraz uwarunkowania, jakie należy uwzględnić podczas ich wykonania

1.8 Ocena kryteriów wyjścia i raportowanie

TA-1.8.1 (K2) Kandydat potrafi wyjaśnić, dlaczego dokładne informacje o statusie wykonania przypadków testowych są istotne

1.9 Czynności związane z zakończeniem testów

TA-1.9.1 (K2) Kandydat potrafi podać przykłady produktów, jakie powinny zostać dostarczone przez Analityka Testów w ramach czynności związanych z zakończeniem testów

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

1.1 Wprowadzenie

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

- planowanie, monitorowanie i nadzór,
- analiza i projektowanie,
- implementacja i wykonanie,
- ocena kryteriów wyjścia i raportowanie,
- czynności związane z zakończeniem testów.

Na poziomie zaawansowanym niektóre z tych elementów rozważa się w rozbiciu na poszczególne czynności tak, aby uszczegółowić i zoptymalizować procesy, lepiej dopasować proces testowy do cyklu życia wytwarzania oprogramowania i zapewnić efektywne monitorowanie testów i nadzór nad nimi. Na tym poziomie rozróżnia się następujące elementy:

- planowanie, monitorowanie i nadzór,
- analiza,
- projektowanie,
- implementacja,
- wykonanie,
- ocena kryteriów wyjścia i raportowanie,
- czynności związane z zakończeniem testów.

Te czynności można wykonywać sekwencyjnie lub częściowo równolegle, np. można projektować testy jednocześnie z ich implementowaniem (jak w przypadku testowania eksploracyjnego). Określenie właściwych testów i przypadków testowych, ich zaprojektowanie i wykonanie to najważniejsze obszary działalności Analityka Testów. Zrozumienie pozostałych kroków procesu testowego jest istotne, ale praca Analityka Testów obejmuje przede wszystkim czynności związane z analizą, projektowaniem, implementacją i wykonaniem w ramach testów.

Doświadczeni testerzy napotykają w swoich organizacjach, zespołach i wykonywanych zadaniach szereg wyzwań przy wdrażaniu różnych aspektów testowania opisanych w niniejszym sylabusie. Należy zawsze uwzględniać różne cykle życia wytwarzania oprogramowania oraz rodzaje testowanych systemów, ponieważ te czynniki mogą wpływać na podejście testowe.

1.2 Testowanie w cyklu życia oprogramowania

W ramach strategii testowania należy rozważyć i zdefiniować długofalowe podejście testowe w cyklu życia oprogramowania. Między modelami cyklu życia oprogramowania istnieją różnice co do momentu zaangażowania Analityka Testów, stopnia tego zaangażowania, wymaganego od Analityka Testów nakładu czasu, dostępnych dla niego informacji i wreszcie oczekiwań względem niego. Proces testowy nie jest odseparowany od innych procesów, więc Analityk Testów musi znać punkty, w jakich następuje przepływ informacji między obszarami organizacji, takimi jak:

- inżynieria wymagań i zarządzanie wymaganiami — przeglądy wymagań,
- zarządzanie projektem — informacje wejściowe do harmonogramów,
- zarządzanie konfiguracją i zarządzanie zmianami — weryfikacja wersji poprzez testowanie, kontrola wersji,
- rozwój oprogramowania — przewidywanie, co i kiedy będzie dostępne,
- pielęgnacja oprogramowania — zarządzanie defektami, czas obróbki (tj. czas od wykrycia defektu do jego rozwiązania),
- wsparcie techniczne — dokładne dokumentowanie sposobów ominięcia defektów,
- tworzenie dokumentacji technicznej (np. specyfikacji projektu bazy danych) — informacje wejściowe do tych dokumentów oraz przeglądy techniczne dokumentów.

Czynności związane z testowaniem muszą być dostosowane do wybranego modelu cyklu życia wytwarzania oprogramowania, który może mieć charakter sekwencyjny, iteracyjny lub przyrostowy. Na

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

przykład w sekwencyjnym modelu V podstawowy proces testowy ISTQB® zastosowany na poziomie testów systemowych można dopasować do procesu rozwoju oprogramowania następująco:

- Planowanie testów systemowych odbywa się równolegle z planowaniem projektu, a nadzór nad testami jest prowadzony do czasu, gdy testy systemowe zostaną wykonane i zamknięte.
- Analiza i projektowanie testów systemowych odbywają się równolegle ze specyfikowaniem wymagań, specyfikowaniem projektu systemu i architektury (specyfikacje wysokiego poziomu) i specyfikowaniem projektów komponentów (specyfikacje niskiego poziomu).
- Implementacja środowiska testowego dla testów systemowych (np. łoże testowe, osprzęt testowy) może się zacząć 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 decyzji o pominięciu sprawdzania) kryteriów wejścia testów systemowych, co zwykle oznacza, że co najmniej testy modułowe, a często również testy integracyjne, zostały zakończone. Wykonywanie testów systemowych trwa do chwili spełnienia kryteriów wyjścia testów systemowych.
- Przez cały czas wykonywania testów systemowych podlega ocenie spełnienie kryteriów wyjścia testów systemowych, a rezultaty tych testów — raportowaniu, przy czym częstotliwość i pilność tych działań zwiększa się wraz ze zbliżaniem się dat granicznych harmonogramu.
- Czynności związane z zakończeniem testów są wykonywane po spełnieniu kryteriów wyjścia testów systemowych i ogłoszeniu zakończenia wykonywania tych testów; czasem odkłada się je jednak na czas po zakończeniu testów akceptacyjnych i wszystkich czynności projektowych.

W modelach iteracyjnych i przyrostowych kolejność działań może być inna, a niektóre z nich mogą się w ogóle nie pojawiać. Na przykład w modelu iteracyjnym w poszczególnych iteracjach może znaleźć zastosowanie tylko część standardowego procesu testowania. Analiza i projektowanie, implementacja i wykonanie oraz ocena i raportowanie mogą być prowadzone w każdej iteracji, natomiast planowanie odbywa się na początku projektu, a raportowanie końcowe — na końcu. W projekcie zwinnym zazwyczaj obowiązują mniej sformalizowane procesy i bliższe kontakty robocze, dzięki czemu wprowadzanie zmian w projekcie jest łatwiejsze. Ponieważ proces zwinny jest „procesem lekkim”, dokumentacja testów jest mniej szczegółowa, lecz w zamian wykorzystuje się szybsze metody komunikacji, takie jak codzienne spotkania na stojąco (nazywane tak, ponieważ trwają bardzo krótko, zwykle ok. 10-15 minut, więc uczestnicy nie muszą siadać i pozostają skoncentrowani).

Spośród wszystkich modeli cyklu życia oprogramowania projekty zwinne wymagają najwcześniejszego zaangażowania Analityka Testów. Analityk Testów powinien oczekiwać pierwszych zadań w fazie inicjowania projektu, kiedy to współpracuje z programistami na wczesnym etapie koncepcji architektury i projektowania. Przeglądy raczej nie są sformalizowane, ale odbywają się w trybie ciągłym wraz z ewolucją oprogramowania. Od Analityka Testów oczekuje się zaangażowania w prace i dostępności dla zespołu przez cały czas trwania projektu. Ze względu na tę intensywność współpracy członkowie zespołów zwinnych zwykle są przydzieleni tylko do jednego projektu i zaangażowani we wszystkie jego aspekty.

Modele iteracyjne/przyrostowe pokrywają szeroki zakres od podejścia zwinnego, w którym oczekuje się ciągłych zmian wraz z ewolucją oprogramowania, po modele iteracyjne/przyrostowe w ramach modelu V (czasem nazywane: z wbudowaną iteracyjnością). We modelu z wbudowaną iteracyjnością Analityk Testów jest zazwyczaj zaangażowany w standardowe działania planowania i projektowania, a podczas prac programistycznych, testowania, modyfikacji i wdrażania oprogramowania zakres jego współpracy z zespołem zwiększa się.

Niezależnie od stosowanego modelu cyklu życia oprogramowania Analityk Testów powinien rozumieć oczekiwania co do czasu i stopnia jego zaangażowania w projekt. W praktyce wykorzystuje się wiele modeli hybrydowych, takich jak wspomniany wyżej wbudowany model iteracyjny w modelu V. Analityk Testów często sam musi określić najbardziej efektywną dla siebie rolę i działać na rzecz jej realizowania, a nie zdawać się na definicję ustalonego modelu wskazującego moment zaangażowania się w projekt.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

1.3 Planowanie, monitorowanie i nadzór nad testami

Tematem tej sekcji są procesy planowania i monitorowania testów oraz nadzoru nad testami.

1.3.1 Planowanie testów

Prace związane z planowaniem testów przypadają przede wszystkim na okres inicjacji testów i obejmują identyfikowanie i planowanie wszystkich czynności i zasobów wymaganych do zrealizowania misji i celów określonych w strategii testów. Podczas planowania testów Analityk Testów we współpracy z Kierownikiem Testów powinien uwzględnić następujące czynności i odpowiednio do nich zaplanować prace:

- Upewnić się, że plany testów nie ograniczają się tylko do testów funkcjonalnych. W planie testów należy uwzględnić wszystkie typy testów i odpowiednio rozplanować harmonogram. Na przykład oprócz testów funkcjonalnych Analityk Testów może być odpowiedzialny za testy użyteczności. W dokumencie planu testów musi wtedy być ujęty również ten typ testów.
- Dokonać przeglądu oszacowań dotyczących testowania z Kierownikiem Testów i zapewnić odpowiednią ilość czasu na pozyskanie i walidację środowiska testowego.
- Zaplanować testy konfiguracji. Jeżeli z różnego typu procesorów, systemów operacyjnych, maszyn wirtualnych, przeglądarek i urządzeń peryferyjnych można zbudować wiele różnych konfiguracji dla testowanego oprogramowania, należy zaplanować zastosowanie technik testowania, które umożliwią odpowiednie pokrycie tych kombinacji.
- Zaplanować testy dokumentacji. Użytkownicy otrzymują zarówno oprogramowanie, jak i dokumentację. Aby dokumentacja była przydatna, musi być dokładna. Analityk Testów musi przewidzieć czas na weryfikację dokumentacji; czasem konieczna jest współpraca z redaktorami technicznymi w celu przygotowania danych użytych do zrzutów ekranu i nagrań.
- Zaplanować testy procedur instalacji. Należy w odpowiednim stopniu przetestować procedury instalacji, a także procedury wykonywania i przywracania kopii zapasowych. Te procedury mogą się okazać istotniejsze niż samo oprogramowanie; jeżeli oprogramowania nie da się zainstalować, nie będzie ono w ogóle używane. Ten element może być trudny do zaplanowania, gdyż Analityk Testów ma do czynienia z testami wstępnymi w prekonfigurowanym systemie, który nie obsługuje jeszcze ostatecznych procesów instalacji.
- Zaplanować testowanie tak, aby przebiegało zgodnie z cyklem życia oprogramowania. Sekwencyjne wykonywanie zadań rzadko odpowiada rzeczywistemu harmonogramowi. Wiele zadań trzeba wykonywać co najmniej częściowo równolegle. Analityk Testów musi wiedzieć, jaki cykl życia oprogramowania ma zastosowanie i jakie są oczekiwania względem zaangażowania Analityka Testów w fazach projektowania, tworzenia i implementacji oprogramowania. Należy także przewidzieć czas na testy potwierdzające i regresywne.
- Uwzględnić odpowiednio dużo czasu na identyfikowanie i analizowanie czynników ryzyka we współpracy z zespołem realizującym wiele funkcji. Analityk Testów nie jest co prawda zwykle odpowiedzialny za zorganizowanie sesji zarządzania ryzykiem, ale powinien spodziewać się czynnego zaangażowania w zadania z nim związane.

Podstawa testów, warunki testowe i przypadki testowe mogą być powiązane złożonymi relacjami wiele-do-wielu. Analityk Testów musi rozumieć te relacje, aby efektywnie zaplanować i nadzorować testy; jest też zwykle najlepszą osobą do ustalenia tych relacji i maksymalnego rozdzielenia tych zależności.

1.3.2 Monitorowanie i nadzór nad testami

Monitorowanie testów i nadzór nad nimi są zwykle zadaniami Kierownika Testów, ale Analityk Testów dostarcza metryki, które umożliwiają taki nadzór.

W ciągu całego cyklu rozwoju oprogramowania należy gromadzić różnego rodzaju dane ilościowe (np. procent ukończenia działań planistycznych, uzyskane procentowe pokrycie, liczba przypadków testowych zakończonych pomyślnie lub niepomyślnie). Dla każdej z tych metryk musi być zdefiniowana podstawa (standard odniesienia), w stosunku do której jest następnie monitorowany postęp. Kierownik Testów zajmuje się opracowaniem informacji o metrykach i raportowaniem sumarycznym, natomiast Analityk

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Testów gromadzi informacje o każdej z metryk. Każdy wykonany przypadek testowy, każde zgłoszenie defektu, każdy osiągnięty kamień milowy muszą być uwzględnione w ogólnych metrykach projektu. Istotne jest wprowadzanie jak najdokładniejszych informacji do wszelkiego rodzaju narzędzi monitorowania, tak aby metryki odzwierciedlały rzeczywisty stan projektu.

Dokładne wartości metryk umożliwiają Kierownikom Testów zarządzanie projektem (śledzenie) i w razie potrzeby inicjowanie zmian (nadzór). Na przykład duża liczba defektów zgłaszanych w konkretnym obszarze oprogramowania może wskazywać na konieczność wykonania dodatkowych testów tego obszaru. Do ustalenia priorytetów pozostałych prac i do przydziału zasobów można się wówczas posłużyć informacjami o pokryciu wymagań i czynników ryzyka (informacjami śledzenia). Informacje o podstawowych przyczynach problemów służą do określania obszarów, w których konieczne jest udoskonalenie procesów. Jeżeli zbierane dane są dokładne, umożliwiają nadzór projektu i dostarczanie jego interesariuszom dokładnych informacji o jego stanie. Uwzględniając dane zgromadzone we wcześniejszych projektach, można lepiej zaplanować kolejne projekty. Dokładne dane mają wiele zastosowań. Jednym z zadań Analityka Testów jest zapewnienie, by dane były dokładne, dostarczone na czas i obiektywne.

1.4 Analiza testów

W fazie planowania testów zostaje zdefiniowany zakres projektu testowego. Na podstawie tej definicji zakresu Analityk Testów:

- analizuje podstawę testów,
- identyfikuje warunki testowe.

Aby Analityk Testów mógł efektywnie dokonać analizy testów, powinny być spełnione następujące kryteria wejścia:

- Istnieje dokument opisujący przedmiot testów, który może służyć za podstawę testów.
- Ten dokument przeszedł przegląd z wystarczająco dobrym wynikiem i został odpowiednio zmodyfikowany po przeglądzie.
- Do dyspozycji są odpowiednie środki i wystarczająca ilość czasu, aby wykonać pozostałe prace związane z testowaniem danego przedmiotu testów.

Warunki testowe identyfikuje się z reguły poprzez analizę podstawy testów i celów testowania. W pewnych sytuacjach, gdy dokumentacja jest nieaktualna lub nie istnieje, warunki testowe można ustalić poprzez rozmowy z odpowiednimi interesariuszami (np. w formie warsztatów lub podczas planowania iteracji w projekcie zwinnym). Na podstawie tych warunków i za pomocą technik projektowania testów określonych w strategii testów i/lub w planie testów ustala się następnie, co należy przetestować.

Warunki testowe są zwykle specyficzne dla danego przedmiotu testów, Analityk Testów powinien jednak uwzględnić pewne standardowe uwarunkowania:

- Warto zazwyczaj zdefiniować 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.
- Jeżeli zdefiniowano elementy ryzyka produktowego, należy zidentyfikować warunki testowe dotyczące każdego z czynników ryzyka i powiązać je z odpowiednimi czynnikami.

Po zakończeniu analizy testów Analityk Testów powinien wiedzieć, jakie dokładnie testy należy zaprojektować, aby zrealizować cele określone dla danego obszaru projektu testowego.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

1.5 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. Na projektowanie testów składają się następujące czynności:

- Określenie, dla których obszarów testowych odpowiednie byłyby przypadki testowe niskiego poziomu (konkretne), a dla których przypadki testowe wysokiego poziomu (logiczne).
- Określenie technik projektowania przypadków testowych, które zapewnią odpowiednie pokrycie testowe,
- Utworzenie przypadków testowych weryfikujących zidentyfikowane warunki testowe.

W całym procesie od analizy i projektowania po implementację i wykonanie 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.

Podczas projektowania testów należy pamiętać o następujących kwestiach:

- Dla niektórych przedmiotów testów lepiej sprawdza się zdefiniowanie tylko warunków testowych, bez schodzenia na poziom udokumentowanych testów. W takich przypadkach należy zdefiniować warunki testowe jako wytyczne do testowania nieudokumentowanego.
- Należy jasno określić kryteria zaliczenia i niezaliczenia 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 cel testowania i względną ważność testu.
- Testy muszą być również zrozumiałe dla innych interesariuszy, którzy będą dokonywać przeglądów testów, takich jak programiści, oraz dla audytorów, których akceptacja może być wymagana.
- Testy należy projektować tak, aby pokrywały wszelkie interakcje oprogramowania z aktorami (np. użytkownikami końcowymi, innymi systemami), a nie tylko interakcje poprzez interfejs widoczny dla użytkownika. Komunikacja między procesami, operacje wsadowe i inne przerwania również powodują interakcje z oprogramowaniem i mogą zawierać defekty, toteż Analityk Testów musi zaprojektować odpowiednie testy, które złagodzą związane z tym ryzyko.
- Testy należy projektować tak, aby przetestować interfejsy między poszczególnymi przedmiotami testów.

1.5.1 Przypadki testowe konkretne i logiczne

Jednym z zadań Analityka Testów jest ustalenie typów przypadków testowych najbardziej odpowiednich do danej sytuacji. Konkretne przypadki testowe zawierają wszystkie szczegółowe informacje i procedury potrzebne testerowi do wykonania przypadku testowego (w tym wymagania dotyczące danych) i do zweryfikowania jego rezultatu. Konkretne przypadki testowe są przydatne, gdy wymagania są dobrze zdefiniowane, testerzy niezbyt doświadczeni albo, gdy jest wymagana zewnętrzna weryfikacja testów, np. w postaci audytu. Konkretne przypadki testowe bardzo łatwo powtórzyć (inny tester uzyska ten sam rezultat), ale mogą one wymagać dużego nakładu pracy przy utrzymaniu i ograniczają swobodę testerów podczas wykonywania testów.

Logiczne przypadki testowe zawierają wskazówki, co ma zostać przetestowane, ale umożliwiają również Analitykowi Testów zastosowanie różnych zestawów danych, a nawet procedur wykonania testu. Logiczne przypadki testowe mogą zapewnić lepsze pokrycie testami niż przypadki konkretne, ponieważ przy każdym wykonaniu będą nieco inne. To jednak powoduje ich mniejszą powtarzalność. Logiczne przypadki testowe sprawdzają się najlepiej, gdy wymagania nie są dobrze zdefiniowane, gdy Analityk Testów, który będzie wykonywać test, ma doświadczenie zarówno w testowaniu, jak i w pracy z produktem, lub gdy formalna dokumentacja nie jest wymagana (np. nie są przeprowadzane audyty testów). Logiczne przypadki testowe można tworzyć wcześnie w procesie gromadzenia wymagań, gdy te ostatnie nie są jeszcze dobrze zdefiniowane. Te przypadki testowe mogą później posłużyć do tworzenia konkretnych przypadków

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

testowych, gdy wymagania już zostaną ustabilizowane i dopracowane. W takiej sytuacji tworzenie przypadków testowych odbywa się sekwencyjnie z przejściem od przypadków logicznych do konkretnych, a do wykonania testów używane są tylko przypadki konkretne.

1.5.2 Tworzenie przypadków testowych

Projektowanie przypadków testowych polega na stopniowym uszczegóławianiu i doprecyzowywaniu zidentyfikowanych warunków testowych zgodnie z technikami projektowania testów (patrz rozdział 3) wskazanymi w strategii testów i/lub w planie testów. Przypadki testowe powinny być powtarzalne, weryfikowalne oraz posiadać dające się prześledzić powiązania z podstawą testów (np. z wymaganiami) zgodnie ze stosowaną strategią testów.

W ramach projektowania przypadków testowych należy zidentyfikować następujące elementy:

- cel,
- warunki wstępne, takie jak wymagania projektowe lub wymagania zlokalizowanego środowiska testowego, plan ich dostarczenia, stan systemu 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,
- warunki wyjściowe, takie jak zmodyfikowane dane, na które ma wpływ wykonanie przypadku testowego, stan systemu, wyzwalacze dalszego przetwarzania itp.

Przed przystąpieniem do tworzenia przypadków testowych należy określić poziom ich szczegółowości, który ma wpływ zarówno na koszt ich opracowania, jak i na stopień powtarzalności podczas wykonywania. Mniej szczegółowe przypadki umożliwiają Analitykowi Testów większą elastyczność podczas ich wykonywania i badanie potencjalnie interesujących obszarów oprogramowania. Z drugiej strony mniejsza szczegółowość zmniejsza zarazem powtarzalność przypadków testowych.

Szczególną trudność może często sprawiać 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, tester powinien odnieść się nie tylko do zawartości wyświetlanej na ekranie, ale również do stanu wyjściowego danych i środowiska. Przy jasno zdefiniowanej podstawie testów wyznaczenie prawidłowych rezultatów teoretycznie powinno być łatwe. Podstawy testów są jednak z reguły nieprecyzyjne, sprzeczne, nie pokrywają kluczowych obszarów przedmiotu testów lub w ogóle nie są dostępne. W takich sytuacjach Analityk Testów musi posiadać odpowiednią wiedzę dziedzinową lub otrzymać wsparcie kogoś, kto ją posiada. Jednak nawet wtedy, gdy podstawa testów jest dobrze określona, zdefiniowanie oczekiwanych rezultatów mogą utrudnić złożone interakcje złożonych zdarzeń wejściowych i reakcji systemu — nieodzowna jest więc wyrocznia testowa. Wykonywanie przypadków testowych bez możliwości sprawdzenia poprawności rezultatów nie niesie ze sobą korzyści, a często powoduje mylne zgłaszanie awarii lub mylne przekonanie o prawidłowym działaniu systemu.

Powyższe czynności mają zastosowanie na wszystkich poziomach testów, choć podstawa testów jest w każdej sytuacji inna. Na przykład testy akceptacyjne wykonywane przez użytkowników mogą opierać się przede wszystkim na specyfikacji wymagań, przypadkach użycia i zdefiniowanych procesach biznesowych, a testy modułowe — na specyfikacjach projektów niskiego poziomu, historyjkach użytkownika i samym kodzie programu. Należy pamiętać, że te czynności są wykonywane na wszystkich poziomach testowania, choć cel testu jest w każdej sytuacji inny. Na przykład testy funkcjonalne na poziomie modułów projektowane są tak, aby zweryfikować, że dany moduł udostępnia funkcjonalność zgodną z jego projektem szczegółowym. Testy funkcjonalne na poziomie integracji weryfikują interakcje między modułami i funkcjonalności dostarczane poprzez interakcje. Celem testów na poziomie systemu powinna być całość funkcjonalności. 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).

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

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

- ryzyko projektowe (co musi / nie musi być udokumentowane),
- „wartość dodana" dokumentacji w projekcie,
- standardy i uregulowania prawne, jakie muszą zostać dotrzymane,
- zastosowany model cyklu życia oprogramowania (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 25000 [ISO25000] (który zastąpił standard ISO 9126) stanowi tu przydatny materiał pomocniczy. Przy testowaniu systemów sprzętowo-programowych może być konieczne uwzględnienie dodatkowych charakterystyk.

Procesy analizy i projektowania testów można udoskonalić poprzez wplecenie w nie przeglądów i testowania statycznego. Tak naprawdę analiza i projektowanie testów same w sobie często stanowią pewną formę testowania statycznego, ponieważ umożliwiają wykrycie problemów w dokumentach bazowych. Analiza i projektowanie testów w oparciu o specyfikację wymagań są znakomitym przygotowaniem do przeglądu wymagań. Lektura wymagań niezbędna do opracowania testów wymaga: zrozumienia wymagania i możliwości ustalenia sposobu sprawdzenia, czy to wymaganie jest spełnione. Dzięki temu często można wychwycić wymagania, które są niejasne, nietestowalne lub dla których nie są zdefiniowane kryteria akceptacji. Również produkty testowania, takie jak przypadki testowe, analiza ryzyk i plany testów, powinny być poddawane przeglądom.

W niektórych projektach, na przykład prowadzonych w modelu zwinnym, wymagania mogą być udokumentowane tylko w minimalnym stopniu. Mogą na przykład mieć postać tzw. historyjek użytkownika, które opisują niewielkie, ale możliwe do zaprezentowania wycinki funkcjonalności. Historyjka użytkownika powinna zawierać kryteria akceptacji. Jeżeli można zademonstrować, że oprogramowanie spełnia te kryteria akceptacji, uznaje się je z reguły za gotowe do zintegrowania z pozostałymi gotowymi elementami funkcjonalnymi lub zostało ono już zintegrowane, aby można było dokonać takiej demonstracji.

Podczas projektowania testów można szczegółowo zdefiniować wymagania dotyczące infrastruktury testowej, w praktyce jednak ich ostateczna wersja powstaje podczas implementacji testów. Należy pamiętać, że infrastruktura testowa 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 tych dwóch sekcjach. Istotne jest, aby kryteria te były mierzalne i zapewniały dostarczenie wszystkich informacji i dokonanie wszystkich przygotowań niezbędnych do wykonania kolejnych kroków w procesie.

1.6 Implementacja testów

Implementacja testów oznacza praktyczną realizację projektowania testów. Ten krok obejmuje stworzenie testów automatycznych, ustalenie kolejności wykonania testów (zarówno manualnych, jak i automatycznych), zakończenie prac nad danymi testowymi i środowiskami testowymi oraz sformułowanie harmonogramu wykonywania testów wraz z przydzieleniem zasobów, tak aby było możliwe rozpoczęcie wykonywania przypadków testowych. W tej fazie zostają również sprawdzone jawne i wywnioskowane kryteria wejścia testów na danym poziomie oraz spełnienie kryteriów wyjścia poprzednich kroków procesu. Pominięcie sprawdzenia kryteriów wyjścia poziomu testów lub kroku w procesie z dużym prawdopodobieństwem będzie oznaczać opóźnienia w realizacji harmonogramów, obniżenie jakości i

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

nieoczekiwane dodatkowe nakłady pracy. Istotne jest, aby zapewnić spełnienie wszystkich kryteriów wyjścia przed rozpoczęciem prac nad implementacją testów.

Przy ustalaniu kolejności wykonywania testów należy uwzględnić różne czynniki. W pewnych przypadkach warto zgrupować przypadki testowe w zestawach testowych. Umożliwia to zorganizowanie testów w taki sposób, aby powiązane ze sobą przypadki testowe były wykonywane łącznie. W przypadku zastosowania strategii testów opartej na ryzyku kolejność wykonywania przypadków testowych może być podyktowana priorytetami czynników ryzyka. Kolejność może również zależeć od innych czynników, takich jak dostępność odpowiedniego personelu, sprzętu, danych i testowanych funkcjonalności. Kod programu często jest publikowany we fragmentach, w związku z czym należy skoordynować testowanie z kolejnością udostępniania poszczególnych elementów oprogramowania. Szczególnie w przypadku przyrostowych modeli cyklu życia oprogramowania Analityk Testów powinien koordynować swoje prace z działaniami zespołu programistów, tak aby elementy oprogramowania były udostępniane do testów w kolejności umożliwiającej ich przetestowanie. Podczas implementacji testów Analityk Testów powinien zakończyć opracowywanie i potwierdzić kolejność wykonania testów manualnych i automatycznych, starannie sprawdzając ograniczenia, które mogłyby wymusić uruchamianie testów w określonym porządku. Należy udokumentować i sprawdzić zależności.

Poziom szczegółowości przypadków testowych i warunków 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 testy powinny wówczas być zgodne z odpowiednimi standardami, takimi jak regulacja DO-178B/ED 12B amerykańskiego Federalnego Urzędu Lotnictwa [RTCA DO-178B/ED-12B].

Jak wcześniej wspomniano, do testowania potrzebne są zestawy danych testowych, które w pewnych przypadkach mogą być naprawdę duże. Podczas implementacji Analityk Testów tworzy dane wejściowe i dane środowiskowe, które mają zostać załadowane do baz danych i innych tego rodzaju repozytoriów. Analityk Testów tworzy również dane, które będą używane w testach automatycznych sterowanych danymi i w testach manualnych.

Implementacja testów obejmuje także tworzenie środowisk testowych. W tej fazie należy w pełni skonfigurować środowiska i zweryfikować 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 odzwierciedla — o ile to wymagane — środowisko produkcyjne lub środowisko użytkownika końcowego dla potrzeb testów na wyższych poziomach. 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 testerzy muszą potwierdzić dostępność osób odpowiedzialnych za przygotowanie i utrzymanie środowiska testowego, dostępność wszystkich testaliów oraz gotowość narzędzi testowych do użycia. Dotyczy to zarządzania konfiguracją, zarządzania defektami oraz logowania testów i zarządzania nimi. Ponadto Analityk Testów musi zweryfikować procedury gromadzenia danych wykorzystywanych do oceny spełnienia kryteriów wyjścia i do raportowania rezultatów testów.

Przy implementacji testów warto zastosować zrównoważone podejście oparte o ustalenia z fazy planowania testów. Na przykład analityczne strategie testów oparte na ryzyku często łączy się z dynamicznymi strategiami testów. W takich sytuacjach pewną część implementacji testów stanowi przygotowanie testów, w których nie postępuje się zgodnie z wcześniej zdefiniowanymi skryptami (testów nieudokumentowanych).

Testowanie bez dokumentacji nie powinno być ad hocowe ani bezcelowe, ponieważ ich czas i pokrycie może być nieprzewidywalne; należy dla nich przewidzieć konkretne ramy czasowe i przygotować karty opisu testów. Na przestrzeni lat praktycy opracowali szereg technik testowania opartych na doświadczeniu, takich jak ataki usterek, zgadywanie błędów [Myers79] i testowanie eksploracyjne. Te podejścia nie

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

eliminują analizy, projektowania i implementacji testów, ale czynności te są realizowane w dużej mierze podczas wykonywania testów.

Przy wykorzystaniu tych dynamicznych strategii testów rezultaty każdego testu mają wpływ na analizę, projektowanie i implementację kolejnych testów. Te strategie wymagają co prawda mniejszego nakładu pracy i często umożliwiają efektywne wykrywanie defektów, mają jednak również wady: wymagają od Analityka Testów wiedzy eksperckiej, trudno przewidzieć czas ich trwania oraz prześledzić uzyskane pokrycie, a bez wsparcia dobrej dokumentacji lub narzędzi ich powtarzalność spada.

1.7 Wykonanie testów

Wykonanie testu rozpoczyna się, gdy zostanie dostarczony przedmiot testów i zostaną spełnione kryteria wejścia wykonania testu (lub zostanie podjęta decyzja o ich pominięciu). Testy należy wykonywać zgodnie z planem określonym podczas ich implementacji, ale Analityk Testów powinien mieć wystarczająco dużo czasu, aby zapewnić pokrycie dodatkowych interesujących przypadków testowych i zachowań, które obserwowane są podczas testowania (wszelkie awarie wykryte za pomocą środków odbiegających od planu należy udokumentować z uwzględnieniem zmian postępowania w odniesieniu do skryptu przypadku testowego, jakie są konieczne do powtórzenia danej awarii). Takie połączenie technik testowania udokumentowanego i nieudokumentowanego (np. eksploracyjnego) umożliwia skuteczniejsze wykrywanie defektów dzięki dokładniejszemu pokryciu testami oraz uniknięcie paradoksu pestycydów.

Kluczowym elementem wykonywania testów jest porównywanie rezultatów rzeczywistych z oczekiwanymi. Analityk Testów musi poświęcić temu zadaniu uwagę, w przeciwnym razie praca włożona w zaprojektowanie i zaimplementowanie testów pójdzie na marne, gdy awarie zostaną przeoczone (rezultat fałszywie negatywny) lub prawidłowe działanie zostanie błędnie sklasyfikowane jako niepoprawne (rezultat fałszywie pozytywny). Gdy rezultat rzeczywisty jest niezgodny z oczekiwanym, występuje incydent. Incydent należy dokładnie przeanalizować, aby ustalić jego przyczynę (którą może, ale nie musi, być defekt przedmiotu testów) i zgromadzić dane potrzebne do jego rozwiązania (więcej szczegółowych informacji o zarządzaniu defektami można znaleźć w rozdziale 6).

Gdy zostanie zidentyfikowana awaria, należy dokładnie sprawdzić dokumentację testów (specyfikację testu, przypadek testowy itp.), aby upewnić się, że jest ona poprawna. Istnieje wiele możliwych przyczyn niepoprawności dokumentacji testów. Jeżeli jest ona niepoprawna, należy ją skorygować i ponownie wykonać test. Zmiany w podstawie testów oraz przedmiocie testów mogą spowodować niepoprawność testu nawet po tym, jak został wielokrotnie pomyślnie wykonany, i testerzy muszą mieć świadomość, że zaobserwowany rezultat może wynikać z takiej niepoprawności.

Podczas wykonywania testów należy odpowiednio logować ich rezultaty. Aby określić faktyczne rezultaty wykonanych testów, których nie zalogowano, może być konieczne ich powtórzenie, co powoduje obniżenie wydajności i opóźnienia w realizacji projektu testowego. Należy zauważyć, że odpowiednie logowanie rezultatów testów zmniejsza problemy związane z pokryciem i powtarzalnością testów w technikach takich jak testowanie eksploracyjne. Przedmiot testów, testalia i środowiska testowe mogą ewoluować, zatem przy logowaniu rezultatów należy uwzględniać konkretne testowane wersje oraz konfiguracje środowiska. Log testów stanowi rejestr chronologiczny istotnych danych szczegółowych dotyczących wykonania testów.

Należy logować rezultaty zarówno poszczególnych testów, jak i działań czy zdarzeń. Podczas wykonywania testów każdy test powinien zostać opatrzony niepowtarzalnym identyfikatorem, a jego status powinien zostać zalogowany. Należy rejestrować wszystkie zdarzenia, jakie mają wpływ na wykonywanie testów, oraz informacje potrzebne do pomiaru pokrycia testami, a także dokumentować przyczyny opóźnień i przerw w testowaniu. Ponadto należy logować informacje potrzebne do nadzoru nad testami, raportowania postępu testów, pomiaru spełnienia kryteriów wyjścia oraz doskonalenia procesu testowego.

Sposób logowania rezultatów zależy od poziomu i strategii testowania. Na przykład w przypadku automatycznych testów modułowych większość informacji powinna być zapisywana poprzez testy automatyczne. W przypadku testowania manualnego Analityk Testów loguje informacje o wykonaniu

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

testów, często za pomocą narzędzia do zarządzania testami, które śledzi wykonanie testów na podstawie tych informacji. Podobnie jak w przypadku implementacji testów, w pewnych sytuacjach ilość logowanych informacji o wykonaniu testów jest podyktowana wymaganiami prawnymi lub wymaganiami audytu.

W niektórych sytuacjach w wykonywaniu testów biorą udział użytkownicy lub klienci. Może to pomóc zbudować u nich zaufanie do systemu, choć wiąże się to z założeniem, że w testach zostanie znalezionych niewiele defektów. Takie założenie jest często błędne na niskich poziomach testowania, ale może być spełnione podczas testów akceptacyjnych.

Podczas wykonywania testów należy uwzględnić m. in. następujące szczególne obszary:

- Zwracać uwagę na „nieistotne" dziwne zdarzenia i badać je dokładniej. Obserwacje lub rezultaty, które wydają się nieistotne, często wskazują na defekty, które (jak góry lodowe) ukryte są pod powierzchnią.
- Sprawdzać, czy produkt nie robi rzeczy, których nie powinien robić. Z zasady głównym celem testowania jest sprawdzanie, czy produkt realizuje zadania, jakie mu postawiono, ale Analityk Testów musi również upewnić się, że produkt nie działa błędnie poprzez wykonywanie dodatkowych, nieoczekiwanych operacji (np. niepożądanych funkcjonalności).
- Być przygotowanym na to, że skonstruowany zestaw testowy będzie się z czasem zmieniać i rozrastać. Kod programów ewoluuje i będzie trzeba zaimplementować dodatkowe testy pokrywające nowe funkcjonalności, a także sprawdzać regresje w innych obszarach oprogramowania. Podczas wykonywania testów często wykrywa się luki w pokryciu. Tworzenie zestawu testowego jest procesem ciągłym.
- Zbierać doświadczenia do wykorzystania w następnych testach. Zadania związane z testowaniem nie kończą się w chwili udostępnienia produktu użytkownikom lub wprowadzenia go do sprzedaży. Najprawdopodobniej powstanie kolejna wersja lub wydanie danego oprogramowania, więc należy zachować wiedzę i przekazać ją testerom odpowiedzialnym za jego dalsze testowanie.
- Nie należy oczekiwać ponownego wykonania wszystkich testów manualnych. Jest to nierealistyczne. Jeżeli Analityk Testów podejrzewa istnienie problemu, powinien go zbadać i odnotować, a nie zakładać, że problem zostanie wychwycony w kolejnej iteracji wykonywania przypadków testowych.
- Poszukiwać dodatkowych przypadków testowych w danych zgromadzonych w narzędziu do śledzenia defektów. Należy rozważyć tworzenie przypadków testowych dla defektów wykrytych podczas testowania nieudokumentowanego lub eksploracyjnego i dodać takie przypadki do zestawu testów regresywnych.
- Znajdować defekty przed testami regresywnymi. Na testy regresywne zwykle pozostaje niewiele czasu i wykrywanie awarii na tym etapie może prowadzić do opóźnień w realizacji harmonogramu. W testach regresywnych zwykle nie znajduje się proporcjonalnie wielu defektów, głównie dlatego, że są to testy, które już wcześniej zostały wykonane (np. na poprzedniej wersji tego samego oprogramowania) i wtedy defekty powinny zostać wychwycone. Nie oznacza to, że należy całkowicie wyeliminować testy regresywne, a jedynie tylko, że ich efektywność w zakresie wykrywania nowych defektów jest mniejsza niż w przypadku innych testów.

1.8 Ocena kryteriów wyjścia i raportowanie

Z punktu widzenia procesu testowego monitorowanie postępu testów oznacza zapewnienie zgromadzenia informacji potrzebnych do zrealizowania wymagań raportowania. Takimi informacjami są na przykład informacje o postępie wykonania testów. Podczas definiowania kryteriów wyjścia w fazie planowania kryteria mogą zostać podzielone na kryteria obowiązkowe („musi" / „nie może") i nieobowiązkowe („powinien" / „nie powinien"). Na przykład kryteria mogą zawierać stwierdzenia, że „nie mogą być otwarte żadne zgłoszenia defektów o priorytecie 1 ani 2" i że „co najmniej 95% wszystkich przypadków testowych powinno być zakończonych pomyślnie". W takiej sytuacji niespełnienie kryterium obowiązkowego powinno spowodować niespełnienie kryteriów wyjścia, natomiast pozytywny wynik 93% wszystkich przypadków testowych może nadal umożliwić przejście do następnej fazy projektu. Kryteria wyjścia muszą być jasno zdefiniowane, aby można było je obiektywnie ocenić.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Analityk Testów odpowiada za dostarczenie informacji wykorzystywanych przez Kierownika Testów do oceny postępu wykonania testów w odniesieniu do kryteriów wyjścia i za zapewnienie dokładności tych danych. Jeżeli, na przykład, system zarządzania testami udostępnia następujące statusy dla wykonanych przypadków testowych:

- zaliczony,
- niezaliczony,
- zaliczony z uwagami,

to Analityk Testów musi jasno określić, co oznaczają poszczególne statusy, i konsekwentnie je stosować. Czy „zaliczony z uwagami" oznacza, że wykryto defekt, ale nie wpływa on na funkcjonalność systemu? Jaki status zostanie nadany w przypadku defektu użyteczności, który powoduje dezorientację użytkownika? Jeżeli udział procentowy zaliczonych przypadków testowych jest obowiązkowym kryterium wyjścia, oznaczenie przypadku testowego jako „niezaliczonego" zamiast „zaliczonego z uwagami" staje się kwestią krytyczną. Należy również uwzględnić przypadki testowe oznaczone jako „niezaliczone", w których przypadku awaria wynika z przyczyny innej niż defekt (np. z niepoprawnej konfiguracji środowiska testowego). Analityk Testów musi wyjaśnić z Kierownikiem Testów wszelkie niejasności co do monitorowanych metryk czy sposobu stosowania statusów, tak aby śledzone informacje były dokładne i spójne w całym w projekcie.

Często zdarza się, że Analityk Testów jest proszony o przygotowanie raportu statusu w trakcie cyklu testowania oraz o dostarczenie informacji do raportu końcowego. Może to obejmować zebranie metryk z systemów zarządzania testami i zarządzania defektami oraz ocenę ogólnego pokrycia i postępu testów. Analityk Testów powinien umieć posługiwać się narzędziami do raportowania i dostarczyć żądane informacje Kierownikowi Testów, który następnie wybierze z nich potrzebne dane.

1.9 Czynności związane z zakończeniem testów

Gdy wykonywanie testów zostanie uznane za zakończone, należy zgromadzić kluczowe dane wyjściowe z prac testowych i albo przekazać je właściwym osobom, albo zarchiwizować. Te zadania składają się na czynności związane z zakończeniem testów. Analityk Testów powinien spodziewać się, że zostanie zaangażowany w dostarczanie produktów testowania osobom, którym są one potrzebne. Na przykład należy poinformować osoby, które będą korzystać z systemu lub wspierać takie korzystanie, o odroczonych lub zaakceptowanych znanych defektach i przekazać testy oraz środowiska testowe osobom odpowiedzialnym za testowanie pielęgnacyjne. Innym produktem testowania może być zestaw testów regresywnych (automatycznych lub manualnych). Informacje o produktach testowania, powiązania między nimi oraz ich powiązania z elementami zewnętrznymi muszą być jasno udokumentowane i muszą do nich zostać nadane odpowiednie prawa dostępu.

Analityk Testów powinien również spodziewać się udziału w spotkaniach retrospektywnych (ang. „lessons learned"), na których dokumentuje się istotne wnioski wynikające zarówno z projektu testowego, jak i z pozostałych elementów cyklu życia wytwarzania oprogramowania, i przygotowuje plany umożliwiające utrwalanie „dobrych" i eliminowanie lub przynajmniej kontrolowanie „złych" praktyk projektowych. Analityk Testów stanowi cenne źródło informacji na takich spotkaniach i musi być w nie zaangażowany, aby można było zebrać informacje przydatne przy udoskonalaniu procesów. Jeżeli tylko Kierownik Testów ma brać udział w takim spotkaniu, Analityk Testów musi przekazać odpowiednie informacje Kierownikowi Testów, tak aby ten mógł zaprezentować rzeczywisty obraz projektu.

Należy również zarchiwizować rezultaty, logi, raporty oraz inne dokumenty i produkty testowania w systemie zarządzania konfiguracją. To zadanie często przypada Analitykowi Testów i jest ważną częścią czynności związanych z zakończeniem testów, zwłaszcza w sytuacjach, gdy te informacje będą potrzebne w przyszłych projektach.

Zwykle to Kierownik Testów decyduje o tym, jakie informacje należy zarchiwizować, jednak Analityk Testów również powinien zastanowić się, jakie informacje byłyby potrzebne, gdyby w przyszłości projekt miał zostać uruchomiony ponownie. Przemyślenie tego na zakończenie projektu może zaoszczędzić wiele miesięcy pracy, gdy projekt zostanie uruchomiony ponownie w przyszłości lub z udziałem innego zespołu.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

2. Zarządzanie testami: zadania Analityka Testów — 90 minut

Słowa kluczowe

ryzyko produktowe, analiza ryzyka, identyfikacja czynników ryzyka, poziom ryzyka, zarządzanie ryzykiem, łagodzenie ryzyka, testowanie oparte na ryzyku, monitorowanie testów, strategia testów

Cele nauczania dotyczące zarządzania testami: zadań Analityka Testów

2.2 Monitorowanie i nadzór nad postępem testów

TA-2.2.1 (K2) Kandydat potrafi wyjaśnić, jakie typy informacji należy śledzić podczas testowania, aby było możliwe odpowiednie monitorowanie projektu oraz nadzór nad nim

2.3 Testowanie rozproszone, zlecone na zewnątrz i zlecone wewnątrz

TA-2.3.1 (K2) Kandydat potrafi podać przykłady dobrych praktyk komunikacyjnych przy testowaniu w środowisku 24-godzinnym

2.4 Zadania Analityka Testów w testowaniu opartym na ryzyku

TA-2.4.1 (K3) Kandydat potrafi brać udział w identyfikowaniu czynników ryzyka dla podanej sytuacji projektowej, dokonać oceny ryzyka i zaproponować odpowiednie środki łagodzące ryzyko

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

2.1 Wprowadzenie

Analityk Testów współpracuje z Kierownikiem Testów w różnych obszarach i dostarcza mu różnego rodzaju danych. Ten rozdział skupia się na tych elementach procesu testowego, w które Analityk Testów wnosi największy wkład. Od Kierownika Testów oczekuje się, że będzie zbierać potrzebne informacje od Analityka Testów.

2.2 Monitorowanie i nadzór nad postępem testów

Postęp testów monitoruje się w pięciu głównych wymiarach:

- ryzyka produktowe (jakościowe),
- defekty,
- testy,
- pokrycie,
- zaufanie.

Ryzyka produktowe, defekty, testy i pokrycie Analityk Testów może mierzyć i raportować w określony sposób podczas projektu lub działania produkcyjnego systemu i często tak właśnie się dzieje. Zaufanie to zwykle subiektywne odczucie, choć można je mierzyć za pomocą ankiet. Gromadzenie informacji o tych metrykach jest jednym z zadań Analityka Testów. Należy pamiętać, że dokładność tych danych ma krytyczne znaczenie, gdyż z niedokładnych danych powstaną niedokładne informacje o trendach, a na ich podstawie mogą zostać wyciągnięte nieodpowiednie wnioski. W najgorszym wypadku niedokładne dane powodują podejmowanie niewłaściwych decyzji zarządczych i szkodzą wiarygodności zespołu testowego.

W przypadku testowania opartego na ryzyku Analityk Testów powinien śledzić:

- które czynniki ryzyka zostały złagodzone w wyniku testowania,
- które czynniki ryzyka nie są złagodzone.

Do śledzenia łagodzenia ryzyka często wykorzystuje się to samo narzędzie, za pomocą którego monitoruje się stopień wykonania testów (np. narzędzie do zarządzania testami). W tym celu zidentyfikowane czynniki ryzyka muszą być odwzorowane na warunki testowe, które z kolei odwzorowane są na przypadki testowe; gdy przypadek testowy zostaje wykonany i zaliczony, ryzyko uznaje się za złagodzone. Dzięki temu informacje o łagodzeniu ryzyka są aktualizowane automatycznie podczas aktualizowania stanu przypadków testowych. Dotyczy to zarówno testów manualnych, jak i automatycznych.

Do śledzenia defektów używa się zwykle wyspecjalizowanego narzędzia. Przy zgłaszaniu defektów dodatkowo rejestrowane są konkretne informacje umożliwiające ich klasyfikowanie. Na podstawie tych informacji generowane są trendy i wykresy przedstawiające postęp testów i jakość oprogramowania. Szczegółowy opis informacji dotyczących klasyfikacji znajduje się w rozdziale 6 Zarządzanie defektami. Stosowany cykl życia oprogramowania może mieć wpływ na ilość gromadzonej dokumentacji defektów i na metody używane do zapisywania informacji o nich.

W miarę wykonywania testów należy zapisywać informacje o statusie przypadków testowych. Zazwyczaj używa się do tego narzędzia do zarządzania testami, ale można to również w razie potrzeby robić ręcznie. Informacje dotyczące przypadku testowego obejmują m. in.:

- status utworzenia przypadku testowego (np. zaprojektowany, po przeglądzie),
- status wykonania przypadku testowego (np. zaliczony, niezaliczony, zablokowany, pominięty),
- informacje o wykonaniu przypadku testowego (np. data i godzina, nazwisko testera, użyte dane),
- artefakty wykonania przypadku testowego (np. zrzuty ekranów, logi z wykonania).

Podobnie jak zidentyfikowane czynniki ryzyka, przypadki testowe należy odwzorować na wymagania, które są przez nie weryfikowane.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Analityk Testów musi pamiętać, że jeżeli przypadek testowy A jest odwzorowany na wymaganie A i jest to jedyny przypadek testowy odwzorowany na to wymaganie, to gdy przypadek testowy A zostanie wykonany i zaliczony, wymaganie A zostanie uznane za spełnione. W praktyce może, ale nie musi tak być. W wielu sytuacjach do starannego przetestowania wymagania potrzebnych jest więcej przypadków testowych, ale ze względu na ograniczenia czasowe tworzy się tylko pewien ich podzbiór. Na przykład jeżeli do starannego przetestowania implementacji wymagania było potrzeba 20 przypadków testowych, ale utworzono i uruchomiono tylko 10, wówczas informacje o pokryciu wymagań wykażą pokrycie na poziomie 100%, podczas gdy w rzeczywistości uzyskano tylko pokrycie na poziomie 50%. Dokładne śledzenie pokrycia oraz śledzenie statusu przeglądu samych wymagań mogą służyć jako miara zaufania.

Ilość (i poziom szczegółowości) zapisywanych informacji zależy od szeregu czynników, w tym od stosowanego cyklu rozwoju oprogramowania. Na przykład w projektach zwinnych rejestruje się zwykle mniej informacji o statusie ze względu na bliską współpracę w zespole i intensywniejszą komunikację bezpośrednią.

2.3 Testowanie rozproszone, zlecone na zewnątrz, zlecone wewnątrz

Nie zawsze wszystkie prace związane z testowaniem są wykonywane przez jeden zespół testowy, złożony ze współpracowników pozostałych uczestników projektu i ulokowany w jednym miejscu razem z pozostałymi członkami zespołu projektowego. Jeżeli prace testowe są prowadzone w wielu lokalizacjach, można je nazwać pracami rozproszonymi. W przypadku jednej lokalizacji można mówić o pracy scentralizowanej. Gdy prace testowe są wykonywane w jednej lub większej liczbie lokalizacji innych niż lokalizacja zespołu projektowego i przez osoby niebędące pracownikami tej samej firmy, co reszta zespołu projektowego, wówczas można nazwać je pracami zleconymi na zewnątrz. Gdy prace testowe są wykonywane w tej samej lokalizacji, co lokalizacja zespołu projektowego, ale przez osoby niebędące pracownikami tej samej firmy, można mówić o pracach zleconych wewnątrz.

W projektach, w których część zespołu testowego jest rozproszona między wieloma lokalizacjami lub wręcz wieloma firmami, Analityk Testów musi poświęcić szczególną uwagę efektywności komunikacji i przekazywania informacji. Niektóre organizacje pracują w modelu „testowania 24-godzinnego”, w którym zespół ulokowany w jednej strefie czasowej przekazuje pod koniec dnia pracę zespołowi z innej strefy czasowej, tak że testy mogą być prowadzone przez całą dobę. Takie podejście wymaga odpowiedniego planowania ze strony Analityka Testów, który przydziela zadania innym i odbiera je od nich. Właściwe zaplanowanie pracy jest istotne w celu rozgraniczenia zakresów odpowiedzialności, ale również dla zapewnienia dostępności potrzebnych informacji.

Jeżeli nie jest możliwa komunikacja ustna, muszą wystarczyć informacje przekazywane pisemnie. Oznacza to konieczność zastosowania poczty elektronicznej, raportów o statusie i efektywnego wykorzystania narzędzi do zarządzania testami i śledzenia defektów. Jeżeli narzędzie do zarządzania testami umożliwia przypisanie poszczególnych testów konkretnym osobom, może służyć również jako narzędzie planowania pracy i ułatwiać przekazywanie zadań między pracownikami. Dokładnie rejestrowane defekty można w razie potrzeby przesyłać do współpracowników do dalszego przetwarzania. Efektywne wykorzystanie tych systemów komunikacji jest centralnym elementem sprawnego działania organizacji, które nie może polegać na bieżących kontaktach osobistych.

2.4 Zadania Analityka Testów w testowaniu opartym na ryzyku

2.4.1 Wprowadzenie

Jednym z zadań Kierownika Testów często jest ogólny nadzór nad ustanowieniem strategii testów opartej na ryzyku i zarządzaniem nią. Kierownik Testów zwykle angażuje Analityka Testów w prace nad zapewnieniem poprawnej implementacji podejścia opartego na ryzyku.

Analityk Testów powinien być czynnie zaangażowany w następujące zadania związane z testowaniem opartym na ryzyku:

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

- identyfikację czynników ryzyka,
- ocenę ryzyka,
- łagodzenie ryzyka.

Te zadania są wykonywane iteracyjnie na przestrzeni całego cyklu projektu, 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.

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 biznesu, jakie mogą dotyczyć projektu, takich jak czynniki ryzyka związane z bezpieczeństwem użytkowania, kwestiami biznesowymi i ekonomicznymi oraz czynnikami politycznymi.

2.4.2 Identyfikacja ryzyka

W procesie identyfikacji czynników ryzyka szanse na wykrycie jak największej liczby potencjalnych istotnych czynników ryzyka są tym większe, im większą próbę interesariuszy w niego zaangażujemy. Analitycy Testów często posiadają unikatową wiedzę o dziedzinie biznesowej testowanego systemu, więc są szczególnie predysponowani do przeprowadzania wywiadów z ekspertami i użytkownikami z danej dziedziny, dokonywania samodzielnych ocen, korzystania z szablonów ryzyka i wspierania innych w korzystaniu z nich, prowadzenia warsztatów dotyczących ryzyka, prowadzenia sesji „burz mózgów" z obecnymi i potencjalnymi użytkownikami systemu, definiowania list kontrolnych do testowania i korzystania z wcześniejszych doświadczeń z podobnymi systemami lub projektami. Analityk Testów powinien przede wszystkim ściśle współpracować z użytkownikami i innymi ekspertami z danej dziedziny, aby ustalić, które obszary ryzyka biznesowego powinny zostać uwzględnione podczas testowania. Analityk Testów może też wnieść duży wkład w identyfikowanie potencjalnego wpływu czynników ryzyka na użytkowników i interesariuszy.

Przykładami czynników ryzyka, jakie mogą zostać zidentyfikowane w projekcie, są:

- problemy z dokładnością funkcjonalności oprogramowania, np. niepoprawne obliczenia,
- problemy z użytecznością, np. brak potrzebnych skrótów klawiszowych,
- problemy z łatwością nauki, np. brak wskazówek dla użytkownika w kluczowych punktach decyzyjnych.

Uwarunkowania dotyczące testowania poszczególnych charakterystyk jakościowych są opisane w rozdziale 4 niniejszego sylabusa.

2.4.3 Ocena ryzyka

Celem identyfikacji czynników ryzyka jest znalezienie jak największej liczby czynników dotyczących danego projektu; przedmiotem oceny ryzyka jest natomiast analiza tych zidentyfikowanych czynników, a konkretnie ich klasyfikacja i ustalenie prawdopodobieństw wystąpienia i wpływu każdego z czynników.

Ustalenie poziomu ryzyka obejmuje z reguły ocenę prawdopodobieństwa wystąpienia danego czynnika ryzyka i jego wpływu w przypadku wystąpienia. Prawdopodobieństwo wystąpienia rozumie się zwykle jako prawdopodobieństwo, że dany potencjalny problem istnieje w testowanym systemie i zostanie zaobserwowany w działaniu produkcyjnym tego systemu. Wywodzi się ono zatem z ryzyka technicznego. Wkład Technicznego Analityka Testów powinien polegać na wyszukiwaniu i ocenianiu poziomów czynników ryzyka technicznego, a wkład Analityka Testów — na ocenianiu potencjalnego wpływu biznesowego wystąpienia danego problemu.

Wpływ w przypadku wystąpienia rozumie się często jako wagę wpływu na użytkowników, klientów lub innych interesariuszy. Wywodzi się on zatem z ryzyka biznesowego. Wkład Analityka Testów powinien polegać na identyfikowaniu i ocenianiu potencjalnego wpływu poszczególnych czynników ryzyka na dziedzinę biznesu lub na użytkowników. Do czynników wpływających na ryzyko biznesowe należą m. in.

- częstość używania danej funkcjonalności,

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

- straty biznesowe,
- potencjalne straty lub obciążenia finansowe, środowiskowe albo społeczne,
- konsekwencje prawne, cywilne lub karne,
- bezpieczeństwo osób i mienia,
- grzywny, utrata licencji,
- brak uzasadnionych technicznie i ekonomicznie sposobów ominięcia problemu,
- widoczność funkcjonalności,
- widoczność awarii, prowadząca do negatywnego rozgłosu i potencjalnych szkód dla reputacji,
- utrata klientów.

Na podstawie dostępnych informacji o ryzyku Analityk Testów wyznacza poziom ryzyka biznesowego zgodnie z wytycznymi otrzymanymi od Kierownika Testów. Poziomy mogą mieć postać słowną (np. niskie, średnie, wysokie) lub liczbową. Nie jest to jednak prawdziwa metryka ilościowa, chyba że istnieje możliwość obiektywnego pomiaru ryzyka według zdefiniowanej skali. Dokładny pomiar prawdopodobieństwa i kosztów/konsekwencji jest zwykle bardzo trudny, więc poziom ryzyka ustala się zwykle jakościowo.

Wartość jakościowa może mieć przypisany symbol liczbowy, ale nie czyni jej to prawdziwą metryką ilościową. Na przykład Kierownik Testów może określić, że ryzyko biznesowe należy klasyfikować za pomocą wartości od 1 do 10, przy czym wartość 1 oznacza największy wpływ na działalność firmy (największe ryzyko). Po przypisaniu poszczególnym czynnikom ryzyka wartości prawdopodobieństwa (oceny ryzyka technicznego) i wpływu (oceny ryzyka biznesowego) można pomnożyć przez siebie te wartości i w ten sposób ustalić ogólną ocenę ryzyka dla danego czynnika. Taka ogólna ocena może następnie posłużyć do ustalenia priorytetów działań łagodzących ryzyko. W niektórych modelach opartych na ryzyku, na przykład PRISMA® [vanVeenendaal12], wartości czynników ryzyka rozważa się osobno, tak aby w podejściu testowym uwzględnić osobno ryzyko biznesowe i techniczne.

2.4.4 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 testowane elementy zaliczyły test czy nie, oraz poprzez uczestnictwo w przeglądach artefaktów związanych z oprogramowaniem, takich jak wymagania, projekty i dokumentacja dla użytkowników.
- Podejmowanie właściwych działań łagodzących ryzyko wskazanych w strategii testów i planie testów.
- Ponowne ocenianie znanych czynników ryzyka w oparciu o dodatkowe informacje zgromadzone w toku projektu, korygowanie wielkości prawdopodobieństwa, wpływu lub obu tych wartości.
- Identyfikowanie nowych czynników ryzyka wykazanych przez informacje zgromadzone podczas testowania.

Testowanie jest formą łagodzenia ryzyka produktowego (jakościowego). Poprzez wykrywanie defektów testerzy zmniejszają ryzyko, informując o istnieniu tych 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 ustalić możliwości łagodzenia ryzyka poprzez badanie możliwości gromadzenia dokładnych danych testowych, tworzenie i weryfikowanie realistycznych scenariuszy użycia oraz prowadzenie lub nadzorowanie badań użyteczności.

2.4.4.1 Ustalanie priorytetów testów

Poziom ryzyka jest również używany 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 mocny zestaw dobrych danych przykładowych, które można przetworzyć i zweryfikować dokładność wyników przetwarzania. Analityk Testów może też stwierdzić, że poważne ryzyko w nowym produkcie stanowią problemy z

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

użytecznością. Zamiast czekać na wykrycie problemów podczas testów akceptacyjnych przez użytkowników Analityk Testów może ustalić wysoki priorytet testów użyteczności w trakcie trwania testów, aby ułatwić zidentyfikowanie i usunięcie problemów z użytecznością już na wczesnym etapie testowania. Priorytety testów należy rozważyć w tak wczesnych fazach planowania jak to możliwe, tak aby dopasować harmonogram 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 wagą ryzyka (podejście „w głąb"); w innych sytuacjach wybiera się próbkę testów dla wszystkich zidentyfikowanych czynników ryzyka, ważoną według wartości ryzyka, aby zapewnić pokrycie każdego czynnika przynajmniej jednym testem (podejście „w szerz").

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

2.4.4.2 Dostosowywanie testów dla potrzeb przyszłych cykli testowania

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

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

W przypadku przydzielenia dodatkowego czasu na testy pokrycie ryzyka można rozciągnąć na obszary o niższym ryzyku.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

3. Techniki testowania — 825 minut

Słowa kluczowe

analiza wartości brzegowych, tworzenie grafów przyczynowo-skutkowych, testowanie w oparciu o listę kontrolną, metoda drzew klasyfikacji, testowanie kombinatoryjne, testowanie w oparciu o tablicę decyzyjną, taksonomia defektów, technika oparta na defektach, analiza dziedzinowa, zgadywanie błędów, klasy równoważności, technika oparta na doświadczeniu, testowanie eksploracyjne, tablica ortogonalna, testowanie tablicy ortogonalnej, testowanie sposobem par, testowanie oparte na wymaganiach, technika oparta na specyfikacji, testowanie przejść pomiędzy stanami, karta opisu testu, testowanie w oparciu o przypadki użycia, testowanie w oparciu o historyjki użytkownika

Cele nauczania dotyczące technik testowania

3.2 Techniki oparte na specyfikacji

TA-3.2.1 (K2) Kandydat potrafi wyjaśnić zastosowanie tworzenia grafów przyczynowo-skutkowych

TA-3.2.2 (K3) Kandydat potrafi stworzyć przypadki testowe dla podanego punktu specyfikacji, korzystając z techniki projektowania testów „klasy równoważności", aby osiągnąć zadany poziom pokrycia

TA-3.2.3 (K3) Kandydat potrafi stworzyć przypadki testowe dla podanego punktu specyfikacji, korzystając z techniki projektowania testów „analiza wartości brzegowych", aby osiągnąć zadany poziom pokrycia

TA-3.2.4 (K3) Kandydat potrafi stworzyć przypadki testowe dla podanego punktu specyfikacji, korzystając z techniki projektowania testów „tablica decyzyjna", aby osiągnąć zadany poziom pokrycia

TA-3.2.5 (K3) Kandydat potrafi stworzyć przypadki testowe dla podanego punktu specyfikacji, korzystając z techniki projektowania testów „testowanie przejść między stanami", aby osiągnąć zadany poziom pokrycia

TA-3.2.6 (K3) Kandydat potrafi stworzyć przypadki testowe dla podanego punktu specyfikacji, korzystając z techniki projektowania testów „testowanie sposobem par", aby osiągnąć zadany poziom pokrycia

TA-3.2.7 (K3) Kandydat potrafi stworzyć przypadki testowe dla podanego punktu specyfikacji, korzystając z techniki projektowania testów „drzewo klasyfikacji", aby osiągnąć zadany poziom pokrycia

TA-3.2.8 (K3) Kandydat potrafi stworzyć przypadki testowe dla podanego punktu specyfikacji, korzystając z techniki projektowania testów „przypadki użycia", aby osiągnąć zadany poziom pokrycia

TA-3.2.9 (K2) Kandydat potrafi wyjaśnić, w jaki sposób historyjki użytkownika są wykorzystywane do prowadzenia testów w projektach zwinnych

TA-3.2.10 (K3) Kandydat potrafi stworzyć przypadki testowe dla podanego punktu specyfikacji, korzystając z techniki projektowania testów „analiza dziedzinowa", aby osiągnąć zadany poziom pokrycia

TA-3.2.11 (K4) Kandydat potrafi przeanalizować system (jego specyfikację wymagań), ustalić, jakie typy defektów zostaną w nim prawdopodobnie znalezione i wybrać odpowiednie techniki projektowania testów oparte na specyfikacji

3.3 Techniki oparte na defektach

TA-3.3.1 (K2) Kandydat potrafi opisać zastosowanie technik testowania opartych na defektach i wskazać różnice między ich zastosowaniem a zastosowaniem technik opartych na specyfikacji

TA-3.3.2 (K4) Kandydat potrafi przeanalizować podaną taksonomię defektów pod kątem możliwości zastosowania w danej sytuacji korzystając z kryteriów dla dobrej taksonomii

3.4 Techniki oparte na doświadczeniu

TA-3.4.1 (K2) Kandydat potrafi opisać zasady technik opartych na doświadczeniu oraz wskazać ich wady i zalety w porównaniu z technikami opartymi na specyfikacji i technikami opartymi na defektach

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

TA-3.4.2 (K3) Kandydat potrafi określić testy eksploracyjne dla podanego scenariusza i wyjaśnić, jak zaraportować rezultaty

TA-3.4.3 (K4) Kandydat potrafi określić dla podanej sytuacji projektowej, jakie techniki oparte na specyfikacji, na defektach lub na doświadczeniu należy zastosować, aby osiągnąć konkretne cele

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

3.1 Wprowadzenie

Techniki projektowania testów opisane w tym rozdziale dzielą się na następujące kategorie:
- oparte na specyfikacji (oparte na zachowaniu, czarnoskrzynkowe),
- oparte na defektach,
- oparte na doświadczeniu.

Te techniki uzupełniają się i w zależności od potrzeb mogą być stosowane na dowolnym poziomie testów.

Należy zauważyć, że wszystkie trzy kategorie można stosować w testowaniu zarówno funkcjonalnych, jak i niefunkcjonalnych atrybutów jakości. Testowanie cech niefunkcjonalnych jest opisane w następnym rozdziale.

Techniki projektowania testów opisane w poniższych sekcjach mają za cel ustalenie optymalnych danych testowych (np. klasy równoważności) lub wyprowadzenie sekwencji testowania (np. modele stanów). Z reguły przy tworzeniu kompletnych przypadków testowych łączy się różne techniki ze sobą.

3.2 Techniki oparte na specyfikacji

Techniki oparte na specyfikacji są stosowane w celu wyprowadzenia przypadków testowych z analizy podstawy testów modułu lub systemu bez odnoszenia się do jego struktury wewnętrznej.

Typowe cechy technik opartych na specyfikacji to m. in.:
- podczas projektowania testów są tworzone modele zgodne z techniką testowania, np. diagramy przejść między stanami i tabele decyzyjne;
- warunki testowe są systematycznie wyprowadzane z tych modeli.

Niektóre techniki dostarczają również kryteria pokrycia, za pomocą których można mierzyć działania związane z projektowaniem i wykonywaniem testów. Całkowite spełnienie kryteriów pokrycia nie oznacza, że został zbudowany pełny zestaw testów, a tylko, że z danego modelu nie można już wywieść więcej testów umożliwiających zwiększenie pokrycia tą techniką.

Testy oparte na specyfikacji bazują zwykle na dokumentacji wymagań systemowych. Specyfikacja wymagań powinna określać zachowanie systemu, zwłaszcza w zakresie funkcjonalnym, więc wyprowadzanie testów z wymagań stanowi zwykle część testowania zachowania systemu. W pewnych sytuacjach dokumentacja wymagań może nie być dostępna, ale istnieją przewidywane wymagania, takie jak zastąpienie funkcjonalności przestarzałego systemu.

Istnieje szereg technik testowania opartych na specyfikacji. Te techniki 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 testami oraz typy defektów, na jakie jest ukierunkowana dana technika.

3.2.1 Klasy równoważności

Klas równoważności używa się do zmniejszenia liczby przypadków testowych wymaganych do efektywnego przetestowania obsługi danych wejściowych, wyjściowych, wartości wewnętrznych i uwarunkowanych czasowo. W procesie podziału na klasy równoważności powstają partycje wartości, które są przetwarzane w taki sam sposób. Przyjmuje się, że w przypadku wybrania jednej reprezentatywnej wartości z danej klasy jest zapewnione pokrycie wszystkich elementów tej klasy.

Obszar zastosowania
Tę technikę można stosować na każdym poziomie testowania, w sytuacjach, gdy wszystkie elementy zbioru wartości do przetestowania mają zostać przetworzone w taki sam sposób i gdy zbiory wartości używanych przez aplikację nie nakładają się na siebie. Dobór zbiorów wartości dotyczy zarówno poprawnych, jak i niepoprawnych wartości (tzn. klasy mogą zawierać wartości, które są uznawane za niepoprawne w

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

przypadku testowanego oprogramowania). Ta technika sprawdza się najlepiej w połączeniu z analizą wartości brzegowych, która rozszerza listę testowanych wartości o wartości brzegowe klas równoważności. Jest to typowa technika używana w testach dymnych nowej wersji lub nowego wydania oprogramowania, ponieważ umożliwia szybkie ustalenie, czy działają podstawowe funkcjonalności.

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, należałoby przetestować 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 testowanym przedmiotem.

Pokrycie
Pokrycie ustala się poprzez podzielenie liczby klas, dla których przetestowano wartość, przez liczbę wszystkich zidentyfikowanych klas. Użycie kilku wartości z jednej klasy nie zwiększa pokrycia.

Typy defektów
Ta technika umożliwia wykrycie defektów funkcjonalnych w obsłudze różnych wartości danych.

3.2.2 Analiza wartości brzegowych

Analiza wartości brzegowych służy do testowania wartości na granicach uporządkowanych klas równoważności. Istnieją dwa sposoby zastosowania tej techniki: testowanie dwóch i trzech wartości. W przypadku testowania dwóch wartości używa się wartości brzegowej (granicznej) i wartości wykraczającej poza wartość brzegową (z najmniejszym możliwym przyrostem). Na przykład jeżeli klasa równoważności zawiera wartości od 1 do 10 z inkrementacją o 0,5, to test dwóch wartości dla górnej granicy uwzględnia wartości 10 i 10,5. Wartości testowane dla dolnej granicy to 1 i 0,5. Granice są określone jako maksymalna i minimalna wartość zdefiniowanej klasy równoważności.

W przypadku testowania trzech wartości używa się wartości tuż poniżej wartości brzegowej, samej wartości brzegowej i wartości tuż powyżej niej. W poprzednim przykładzie test górnej granicy uwzględniałby wartości 9,5, 10 i 10,5, a dla dolnej granicy zostałyby przetestowane wartości 1,5, 1 i 0,5. Wybór między testowaniem dwóch a trzech wartości powinien być podyktowany ryzykiem związanym z testowanym elementem, przy czym podejścia z trzema wartościami używa się w przypadku elementów o wyższym ryzyku.

Obszar zastosowania
Ta technika ma zastosowanie na wszystkich poziomach testowania i można jej użyć, gdy istnieją uporządkowane klasy równoważności. Uporządkowanie jest istotne ze względu na konieczność użycia elementów zbioru leżących na jego granicy i tuż poza tą granicą. Na przykład zakres liczb jest klasą uporządkowaną. Klasa zawierająca wszystkie obiekty prostokątne nie jest klasą uporządkowaną i nie posiada wartości brzegowych. Oprócz zakresów liczb analizę wartości brzegowych można zastosować do następujących elementów:

- atrybuty liczbowe zmiennych innych niż liczbowe (np. długość),
- pętle, w tym pętle w przypadkach użycia,
- składowane struktury danych,
- obiekty fizyczne (w tym pamięć),
- czynności zależne od czasu.

Ograniczenia/trudności
Dokładność tej techniki zależy od dokładnego zidentyfikowania klas równoważności, więc wykazuje ona te same ograniczenia i trudności, co podział na klasy równoważności. Analityk Testów powinien również znać wielkość inkrementacji wartości poprawnych i niepoprawnych, aby móc ustalić odpowiednie wartości do

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

przetestowania. Analizę wartości brzegowych można przeprowadzić tylko z klasami uporządkowanymi, ale nie ogranicza się to tylko do zakresów wartości poprawnych. Na przykład w przypadku testowania 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 (zarówno dla testowania dwóch, jak i trzech wartości). W ten sposób uzyskuje się wartość pokrycia dla testów wartości brzegowych.

Typy defektów
Analiza wartości brzegowych sprawdza się przy wyszukiwaniu błędnie ustawionych lub nieustawionych granic zbiorów, a także umożliwia wykrycie nadmiarowych granic. Za pomocą tej techniki można wykryć defekty związane z przetwarzaniem wartości brzegowych, zwłaszcza błędy użycia operatorów „mniej niż" i „więcej niż” (przesunięcie). Można jej też użyć do wykrywania defektów niefunkcjonalnych, np. związanych z limitami obciążenia (jak system obsługujący 10 000 użytkowników jednocześnie).

3.2.3 Tablice decyzyjne

Tablice decyzyjne służą do testowania współdziałania kombinacji warunków. Stanowią przejrzystą metodę sprawdzenia, że są testowane wszystkie istotne kombinacje warunków i że testowane oprogramowanie obsługuje wszystkie ich możliwe kombinacje. Testowanie przy użyciu tablic decyzyjnych ma zapewnić, ze zostaną przetestowane wszystkie kombinacje warunków, relacji i ograniczeń. Próba przetestowania wszystkich możliwych kombinacji może prowadzić do powstania ogromnych tablic decyzyjnych. Można strategicznie zmniejszyć liczbę kombinacji ze wszystkich możliwych do tylko tych „interesujących” poprzez tzw. testowanie w oparciu o zredukowane tablice decyzyjne. Przy użyciu tej techniki można ograniczyć listę kombinacji do takich, które wygenerują różne sytuacje wyjściowe, usuwając zestawy warunków, które nie mają znaczenia dla rezultatu. Testy powtarzające się i takie, których kombinacje warunków nie są możliwe do uzyskania, zostają usunięte. Decyzję o zastosowaniu pełnych lub zredukowanych tablic decyzyjnych podejmuje się zwykle w oparciu o ryzyko. [Copeland03]

Obszar zastosowania
Tę technikę stosuje się z reguły na poziomach testów integracyjnych, systemowych i akceptacyjnych. W zależności od kodu programu może ona się sprawdzać również w testowaniu modułowym, jeżeli dany moduł zawiera logikę decyzyjną. Technika tablic decyzyjnych jest szczególnie przydatna, gdy wymagania są zapisane w postaci diagramów przepływu lub tabel reguł biznesowych. Tablice decyzyjne są również stosowane do definiowania wymagań i niektóre specyfikacje wymagań są już sformułowane w takiej postaci. Nawet w sytuacjach, gdy wymagania nie mają formy tabeli ani diagramu przepływu, można zwykle ustalić kombinacje warunków na podstawie opisów słownych. Przy projektowaniu tablic decyzyjnych należy uwzględnić zarówno zdefiniowane kombinacje warunków, jak i takie, które nie są jawnie wskazane, ale wystąpią w praktyce. Aby zaprojektować poprawną tablicę decyzyjną, tester musi mieć możliwość wyprowadzenia wszystkich oczekiwanych wyników dla wszystkich kombinacji warunków ze specyfikacji lub przy użyciu wyroczni testowej. Tablica decyzyjna jest dobrym narzędziem projektowania testów tylko wtedy, gdy zostały w niej uwzględnione wszystkie warunki, które wchodzą ze sobą w interakcje.

Ograniczenia/trudności
Ustalenie wszystkich wchodzących ze sobą w interakcje warunków może być trudne, zwłaszcza gdy wymagania nie są dobrze zdefiniowane lub nie istnieją. Często okazuje się, że oczekiwany rezultat dla przygotowanego zestawu warunków nie jest znany.

Pokrycie
Minimalne pokrycie testami w przypadku tablicy decyzyjnej uzyskuje się poprzez przygotowanie co najmniej jednego przypadku testowego dla każdej kolumny. Zakłada się przy tym, że nie istnieją warunki złożone i że w kolumnie uwzględniono wszystkie możliwe kombinacje warunków. Przy ustalaniu listy testów na

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

podstawie tablicy decyzyjnej należy również uwzględnić wszelkie warunki brzegowe, jakie powinny zostać przetestowane. Warunki brzegowe mogą przyczynić się do zwiększenia liczby przypadków testowych potrzebnych do odpowiedniego przetestowania oprogramowania. Techniki analizy wartości brzegowych i klas równoważności uzupełniają technikę tablic decyzyjnych.

Typy defektów
Typowe wykrywane defekty to nieprawidłowe przetwarzanie przy określonych kombinacjach warunków, które dostarcza nieoczekiwane rezultaty. Podczas tworzenia tablic decyzyjnych mogą zostać wykryte defekty dokumentu specyfikacji. Najczęstsze defekty to pominięcia (brak informacji, co powinno się zdarzyć w określonej sytuacji) i sprzeczności. Podczas testowania można również wykryć problemy z kombinacjami warunków, które nie są obsługiwane lub są obsługiwane niewłaściwie.

3.2.4 Tworzenie grafów przyczynowo-skutkowych

Grafy przyczynowo-skutkowe można utworzyć na podstawie dowolnego źródła opisującego logikę funkcjonalną („reguły") programu, na przykład historyjek użytkownika albo diagramów przepływu. Są one przydatne jako graficzny przegląd struktury logicznej programu i zwykle używa się ich jako podstawy do zbudowania tablic decyzyjnych. Ujęcie decyzji w postaci grafów przyczynowo-skutkowych i/lub tablic decyzyjnych umożliwia systematyczne pokrycie testami logiki programu.

Obszar zastosowania
Grafy przyczynowo-skutkowe stosuje się w tych samych sytuacjach i na tych samych poziomach testowania, co tablice decyzyjne. Graf przyczynowo-skutkowy ilustruje w szczególności kombinacje warunków, które powodują określone rezultaty (wyniki) i które wykluczają pewne rezultaty (operator NOT), zestawy warunków, które muszą mieć wartość „prawda", aby pewien rezultat został osiągnięty (operator AND) oraz warunki alternatywne, które mogą mieć wartość „prawda", aby pewien rezultat został osiągnięty (operator OR). Te relacje łatwiej prześledzić na grafie przyczynowo-skutkowym niż w opisie słownym.

Ograniczenia/trudności
W porównaniu z innymi technikami projektowania testów na tworzenie grafów przyczynowo-skutkowych potrzebny jest dodatkowy czas i nakład pracy związany z opanowaniem tej metody. Niezbędne są również odpowiednie narzędzia. W grafach przyczynowo-skutkowych wykorzystuje się ustalony sposób zapisu i zarówno twórca grafu, jak i jego odbiorca muszą go rozumieć.

Pokrycie
W celu osiągnięcia minimalnego pokrycia musi zostać przetestowana każda możliwa ścieżka od przyczyny do skutku łącznie z odpowiednimi kombinacjami warunków. Grafy przyczynowo-skutkowe umożliwiają zdefiniowanie ograniczeń dotyczących danych i przepływu logiki.

Typy defektów
Grafy umożliwiają wykrycie tych samych defektów kombinatoryjnych, co tablice decyzyjne. Ponadto tworzenie grafów ułatwia określenie wymaganego poziomu szczegółowości podstawy testów, a przez to podniesienie szczegółowości i jakości podstawy testów oraz zidentyfikowanie brakujących wymagań.

3.2.5 Testowanie przejść pomiędzy stanami

Testowanie przejść pomiędzy stanami umożliwia weryfikację zdolności oprogramowania do wchodzenia w zdefiniowane stany i wychodzenia z nich poprzez poprawne i niepoprawne przejścia. W reakcji na zdarzenia oprogramowanie zmienia 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.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Do śledzenia przejść pomiędzy stanami używa się albo diagramu przejść pomiędzy stanami, który przedstawia w postaci graficznej wszystkie poprawne przejścia pomiędzy stanami, albo tablicy stanów, która zawiera wszystkie możliwe przejścia, zarówno poprawne, jak i niepoprawne.

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, WWW i transakcyjne, a także systemy sterowania, np. sterowniki sygnalizatorów świetlnych.

Ograniczenia/trudności
Najtrudniejszą częścią definiowania tablicy lub diagramu stanów jest często ustalenie listy stanów. W przypadku oprogramowania z interfejsem użytkownika jako definicji stanów używa się często ekranów wyświetlanych przez aplikację. W oprogramowaniu wbudowanym stany mogą zależeć od stanów elementów sprzętowych.

Oprócz samych stanów podstawową jednostką przejść pomiędzy stanami jest pojedyncze przejście (0-przełączenie), Przy prostym przetestowaniu wszystkich przejść można wykryć pewne defekty przejść pomiędzy stanami, ale większe możliwości daje testowanie sekwencji transakcji/przejść. Sekwencję dwóch następujących po sobie przejść określa się jako 1-przełączenie, sekwencję trzech kolejnych przejść jako 2-przełączenie itp. (Inną nazwą jest przełączenie N-1, gdzie N oznacza liczbę przejść, jakie zostaną wykonane. Na przykład pojedyncze przejście, 0-przełączenie, nazywa się wówczas przełączeniem 1-1). [Bath08]

Pokrycie
Podobnie jak w przypadku innych technik testowania istnieje tu hierarchia poziomów pokrycia testami. Minimalne akceptowalne pokrycie obejmuje odwiedzenie wszystkich stanów i wykonanie wszystkich przejść. Stuprocentowe pokrycie przejść (nazywane też pokryciem 0-przełączeń lub pokryciem gałęzi logicznych) 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 i przejściami może być konieczne wielokrotne wykonanie niektórych przejść, aby wykonać inne przejścia na zależnych od nich ścieżkach.

Pojęcie „pokrycia N-przełączeń” odnosi się do liczby wykonanych przejść. 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” odnosi się do sytuacji, gdy sekwencje przejść tworzą pętle. 100% pokrycie okrążeń oznacza, że przetestowano wszystkie pętle z dowolnego stanu z powrotem do tego samego stanu. Taki test należy wykonać dla wszystkich stanów, przez które przechodzą pętle.

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

Typy defektów
Typowe defekty to niepoprawne przetwarzanie w aktualnym stanie, będące wynikiem przetwarzania w stanie poprzednim, nieprawidłowe lub nieobsługiwane przejścia, stany pozbawione wyjść oraz potrzebne, a nieistniejące stany lub przejścia. Podczas tworzenia modelu maszyny stanów mogą zostać wykryte defekty dokumentu specyfikacji. Najczęstsze defekty to pominięcia (brak informacji, co powinno się zdarzyć w określonej sytuacji) i sprzeczności.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

3.2.6 Techniki testowania kombinatoryjnego

Testowanie kombinatoryjne stosuje się w przypadku oprogramowania, które korzysta z wielu parametrów przyjmujących wiele różnych wartości; liczba kombinacji jest wtedy zbyt duża, aby było możliwe ich przetestowanie w przewidzianym czasie. Parametry muszą być od siebie niezależne i zgodne w takim znaczeniu, że każda opcja każdego z nich może utworzyć kombinację z każdą opcją dowolnego innego parametru. Drzewa klasyfikacji umożliwiają wykluczenie niektórych kombinacji, jeżeli pewne opcje są ze sobą niezgodne. Nie oznacza to, że czynniki połączone w kombinacji nie będą na siebie wzajemnie wpływać; może się tak dziać, ale ten wpływ powinien mieć akceptowalny zakres.

Testowanie kombinatoryjne umożliwia zidentyfikowanie odpowiedniego podzbioru tych kombinacji do osiągnięcia zadanego poziomu pokrycia. Analityk Testów może wybrać liczbę elementów, jakie mają zawierać kombinacje, w tym elementy pojedyncze, pary, trójki i większe [Copeland03]. Istnieje szereg narzędzi wspomagających Analityka Testów w tym zadaniu (przykłady są dostępne pod adresem www.pairwise.org). W tych narzędziach należy albo wprowadzić listę parametrów i ich wartości (testowanie sposobem par i w oparciu o tablice ortogonalne), albo przedstawić je w postaci graficznej (drzewa klasyfikacji) [Grochtmann94]. Testowanie sposobem par to metoda testowania kombinacji par zmiennych. Tablice ortogonalne to predefiniowane, matematycznie dokładne tablice, które umożliwiają Analitykowi Testów podstawienie testowanych elementów pod zmienne w tablicy, a przez to wygenerowanie zestawu kombinacji, które zapewnią określony poziom pokrycia testami [Koomen06]. Drzewo klasyfikacji umożliwia Analitykowi Testów zdefiniowanie wielkości testowanych kombinacji (np. kombinacje 2, 3 wartości itp.).

Obszar zastosowania
Problem zbyt wielu kombinacji wartości parametrów przejawia się w co najmniej dwóch sytuacjach związanych z testowaniem. Niektóre przypadki testowe zawierają po kilka parametrów, każdy z kilkoma możliwymi wartościami — na przykład ekran z kilkoma polami do wprowadzania danych. Kombinacje wartości parametrów stanowią dane wejściowe takich przypadków testowych. Ponadto niektóre systemy można konfigurować w kilku różnych aspektach, co skutkuje potencjalnie zbyt dużą przestrzenią możliwych konfiguracji. W obu tych sytuacjach można posłużyć się techniką testowania kombinatoryjnego, aby wybrać możliwy do przetestowania podzbiór kombinacji.

W przypadku parametrów o dużej liczbie wartości można najpierw zastosować technikę klas równoważności lub inny mechanizm wyboru, aby zmniejszyć liczbę wartości danego parametru, a następnie wykonać testowanie kombinatoryjne, aby zmniejszyć zbiór kombinacji wynikowych.

Te techniki stosuje się zwykle na poziomach testów integracyjnych, systemowych i integracji systemów.

Ograniczenia/trudności
Głównym ograniczeniem tych technik jest założenie, że rezultaty kilku testów są reprezentatywne dla wszystkich testów i że te kilka testów reprezentuje 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.

Czasem trudności sprawia identyfikacja parametrów i ich wartości. Trudno ręcznie ustalić minimalny zestaw kombinacji, który zapewni określony poziom pokrycia, i z reguły używa się do tego specjalnych narzędzi. Niektóre narzędzia obsługują wymuszanie uwzględnienia lub wykluczenia pewnych kombinacji lub ich wariantów 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
Dostępnych jest kilka poziomów pokrycia. Najniższy poziom nosi nazwę pokrycia pojedynczego. Wymaga on, aby każda wartość każdego parametru została użyta w co najmniej jednej z wybranych kombinacji. Kolejny poziom nosi nazwę pokrycia par. W tym przypadku każda para wartości dwóch dowolnych

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

parametrów musi być uwzględniona w co najmniej jednej kombinacji. Można uogólnić tę koncepcję do poziomu pokrycia kombinacji n wartości, który wymagałby uwzględnienie każdej częściowej kombinacji wartości dowolnego zestawu n parametrów w zbiorze wybranych kombinacji. Im wyższa wartość n, tym większa liczba kombinacji jest niezbędna do osiągnięcia 100% pokrycia. Minimalne pokrycie przy tych technikach wymaga, aby dla każdej kombinacji wygenerowanej przez narzędzie został przygotowany jeden przypadek testowy.

Typy defektów
Defekty wykrywane najczęściej w tego rodzaju testach wiążą się z kombinacjami warunków różnych parametrów.

3.2.7 Testowanie w oparciu o przypadki użycia

Testowanie w oparciu o przypadki użycia umożliwia przeprowadzenie transakcyjnych testów opartych na scenariuszach, które powinny naśladować użytkowanie systemu. Przypadki użycia definiują interakcje aktorów z systemem służące osiągnięciu jakiegoś celu. Aktorami mogą być użytkownicy lub systemy zewnętrzne.

Obszar zastosowania
Testowanie w oparciu o przypadki użycia wykonuje się zwykle na poziomie testów systemowych i akceptacyjnych. Może się ono również sprawdzać w testach integracyjnych zależnie od poziomu integracji, a nawet w testach modułowych zależnie od działania danego modułu. Przypadki użycia często stanowią również podstawę testów wydajnościowych, ponieważ odzwierciedlają realistyczne użytkowanie systemu. Scenariusze opisane w przypadkach użycia można przypisać do użytkowników wirtualnych w celu zbudowania realistycznego obciążenia systemu.

Ograniczenia/trudności
Aby przypadki użycia były poprawne, muszą opisywać realistyczne działania użytkownika. Informacje te powinny pochodzić od użytkowników lub ich przedstawicieli. Wartość przypadku użycia zmniejsza się, gdy przypadek ten nie odzwierciedla dokładnie rzeczywistych działań użytkownika. Dla dokładnego pokrycia testami ważne jest dokładne zdefiniowanie ścieżek (przepływów) alternatywnych. Przypadki użycia powinny służyć jako wskazówki, ale nie jako jedyna definicja elementów do przetestowania, ponieważ mogą nie zawierać jasnej definicji wszystkich wymagań. Warto utworzyć na podstawie opisu słownego przypadku użycia również inne modele, na przykład diagramy przepływu, aby zwiększyć dokładność testowania i zweryfikować sam przypadek użycia.

Pokrycie
Do uzyskania minimalnego pokrycia przypadku użycia potrzebujemy jednego przypadku testowego dla ścieżki głównej (pozytywnej) i po jednym przypadku dla każdej ścieżki alternatywnej (przepływu alternatywnego). Ścieżki alternatywne obejmują również ścieżki wyjątków i awarii. Czasem ścieżki alternatywne są prezentowane jako rozszerzenia ścieżki głównej. Procentowe pokrycie oblicza się przez podzielenie liczby przetestowanych ścieżek przez łączną liczbę ścieżek głównych i alternatywnych.

Typy defektów
Wykrywane defekty to: nieprawidłowe przetwarzanie zdefiniowanych scenariuszy, brak obsługi ścieżek alternatywnych, nieprawidłowe przetwarzanie podanych warunków oraz nieprawidłowe lub utrudniające pracę zgłaszanie błędów.

3.2.8 Testowanie w oparciu o historyjki użytkownika

W niektórych metodykach zwinnych, takich jak Scrum, wymagania opracowuje się w postaci historyjek użytkownika, które opisują niewielkie jednostki funkcjonalne, które można zaprojektować, zaprogramować, przetestować i zaprezentować w toku jednej iteracji [Cohn04]. Historyjki użytkownika zawierają opis funkcjonalności, jaką należy zaimplementować, kryteria niefunkcjonalne oraz kryteria akceptacji, jakie muszą być spełnione, aby historyjkę użytkownika można było uznać za zrealizowaną.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Obszar zastosowania
Historyjki użytkownika stosuje się przede wszystkim w środowiskach zwinnych oraz środowiskach iteracyjnych i przyrostowych, zarówno w testach funkcjonalnych, jak i niefunkcjonalnych. Historyjki użytkownika są wykorzystywane na wszystkich poziomach testowania, przy czym od programisty oczekuje się zaprezentowania zaimplementowanej funkcjonalności z historyjki użytkownika przed przekazaniem kodu do członków zespołu zajmujących się testowaniem na następnym poziomie (np. integracyjnym, wydajnościowym).

Ograniczenia/trudności
Ponieważ historyjki są małymi jednostkami funkcjonalnymi, może być konieczne utworzenie sterowników i zaślepek do przetestowania dostarczonego fragmentu oprogramowania. Wymaga to zazwyczaj umiejętności programowania i znajomości narzędzi pomocniczych takich jak narzędzia do testowania interfejsów API. Utworzenie sterowników i zaślepek jest z reguły zadaniem programisty, choć w tworzenie odpowiedniego kodu i zastosowanie narzędzi do testowania interfejsów API może zostać zaangażowany również Techniczny Analityk Testów. W przypadku projektów prowadzonych metodą integracji ciągłej, jak większość projektów zwinnych, zapotrzebowanie na sterowniki i zaślepki jest ograniczone do minimum.

Pokrycie
Minimalne pokrycie historyjki użytkownika uzyskuje się przez zapewnienie, że każde z podanych kryteriów akceptacji jest spełnione.

Typy defektów
Defekty mają zwykle charakter funkcjonalny — oprogramowanie nie realizuje określonej funkcjonalności. Zdarzają się również defekty integracji funkcjonalności nowej historyjki z funkcjonalnością już istniejącą. Historyjki mogą być opracowywane niezależnie od siebie, mogą się więc pojawiać problemy z wydajnością, interfejsami i obsługą błędów. Za każdym razem, gdy nowa historyjka zostaje udostępniona do testowania, Analityk Testów powinien wykonać zarówno testy danej dostarczonej funkcjonalności, jak i testy integracyjne.

3.2.9 Analiza dziedzinowa

Dziedzina jest określonym zbiorem wartości. Ten zbiór może być zdefiniowany jako zakres wartości pewnej zmiennej (dziedzina jednowymiarowa, np. „mężczyźni w wieku powyżej 24 lat i poniżej 66 lat") lub jako zakresy wartości zmiennych współdziałających ze sobą (dziedzina wielowymiarowa, np. „mężczyźni w wieku powyżej 24 lat i poniżej 66 lat ORAZ ważący ponad 69 kg i poniżej 90 kg"). Każdy przypadek testowy dla dziedziny wielowymiarowej musi zawierać odpowiednie wartości dla każdej ze współdziałających zmiennych.

W przypadku dziedzin jednowymiarowych do analizy dziedzinowej stosuje się zwykle podział na klasy równoważności i analizę wartości brzegowych. Po zdefiniowaniu klas Analityk Testów wybiera dla każdej klasy wartości: zawartą w tej klasie (IN), spoza tej klasy (OUT), na granicy klasy (ON) i tuż poza granicą klasy (OFF). Ustalenie tych wartości umożliwia przetestowanie każdej z klas wraz z warunkami brzegowymi. [Black07]

W przypadku dziedzin wielowymiarowych liczba przypadków testowych utworzonych w ten sposób rośnie wykładniczo wraz z liczbą uwzględnianych zmiennych, jeżeli natomiast użyć zamiast tego metod opartych na teorii dziedzin, ten przyrost przebiega liniowo. Ponadto ponieważ formalne podejście w odróżnieniu od podziału na klasy równoważności i analizy wartości brzegowych uwzględnia teorię defektów (model usterek), mniejsze zestawy testowe umożliwiają znalezienie w dziedzinach wielowymiarowych defektów, których większe zestawy heurystyczne prawdopodobnie by nie wykryły. Dla dziedzin wielowymiarowych model testów może mieć postać tablicy decyzyjnej (albo „macierzy dziedziny"). Do zidentyfikowania wartości przypadków testowych dla dziedzin wielowymiarowych o liczbie wymiarów większej niż 3 jest zwykle potrzebne wsparcie narzędziowe.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Obszar zastosowania
Analiza dziedzinowa łączy w sobie techniki tworzenia tabel decyzyjnych, podziału na klasy równoważności i analizy wartości brzegowych, tak aby zbudować mniejszy zbiór testów, który mimo to pokryje istotne obszary oraz prawdopodobne obszary występowania awarii. Stosuje się ją często w sytuacjach, gdy tablice decyzyjne stałyby się nieporęczne ze względu na dużą liczbę zmiennych, które mogą ze sobą współdziałać. Analizę dziedzinową można wykorzystać na dowolnym poziomie testowania, ale z reguły używa się jej na poziomach testów integracyjnych i systemowych.

Ograniczenia/trudności
Staranna analiza dziedzinowa wymaga dokładnego zrozumienia sposobu działania oprogramowania w celu zidentyfikowania dziedzin i potencjalnych interakcji między nimi. W przypadku pominięcia jakiejś dziedziny testy mogą zawierać poważne luki, ale jest prawdopodobne, że taka dziedzina zostanie wykryta, ponieważ wartości OFF i OUT mogą do niej należeć. Analiza dziedzinowa sprawdza się bardzo dobrze, gdy Analityk Testów współpracuje z programistą przy definiowaniu testowanych obszarów.

Pokrycie
W celu zapewnienia minimalnego pokrycia w technice analizy dziedzinowej niezbędny jest jeden test dla każdej wartości IN, OUT, ON i OFF każdej z dziedzin. W przypadku powtarzających się wartości (np. gdy wartość OUT jednej dziedziny jest wartością IN innej) nie trzeba wykonywać testu dwa razy. Z tego względu rzeczywista liczba potrzebnych testów wynosi zwykle mniej niż 4 na dziedzinę.

Typy defektów
Wykrywane defekty to: problemy funkcjonalne z dziedziną, błędna obsługa wartości brzegowych, problemy związane z interakcjami między zmiennymi oraz obsługa błędów (zwłaszcza w przypadku wartości nienależących do poprawnej dziedziny).

3.2.10 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, pokrywają dodatkowo poszczególne klasy równoważności. Przy podejmowaniu decyzji o zastosowaniu lub nie 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). Może być tak, że w danej sytuacji nie da się wybrać „najlepszej" techniki. Łączenie technik często zapewnia największy stopień pokrycia, o ile zespół testowy dysponuje odpowiednią ilością czasu i/lub umiejętnościami niezbędnymi do prawidłowego zastosowania tego podejścia.

3.3 Techniki oparte na defektach

3.3.1 Korzystanie z technik opartych na defektach

W technikach projektowania testów opartych na defektach punktem wyjścia dla projektu testu jest typ poszukiwanych defektów; testy systematycznie wyprowadza się z wiedzy na temat tego typu defektów. W odróżnieniu od testowania opartego na specyfikacji, którego podstawą jest specyfikacja produktu, w testowaniu opartym na defektach testy są wyprowadzane z taksonomii defektów (np. list podzielonych na kategorie), które mogą być całkowicie niezależne od testowanego oprogramowania. Taksonomia defektów może zawierać listy typów defektów, ich podstawowe przyczyny, objawy awarii i inne dane dotyczące defektów. W ukierunkowanym testowaniu opartym na defektach można również używać list zidentyfikowanych czynników ryzyka oraz scenariuszy ryzyka. Ta technika testowania umożliwia testerowi ukierunkowanie działań na określony typ defektów lub przejście krok po kroku przez taksonomię defektów opisującą znane i częste defekty danego typu. Analityk Testów na podstawie danych taksonomii ustala cel testów, jakim jest wykrycie defektów danego typu. Na podstawie tych informacji Analityk Testów tworzy

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

przypadki testowe i warunki testowe, które powodują ujawnienie tych defektów, jeżeli są one obecne w produkcie.

Obszar zastosowania
Testowanie oparte na defektach sprawdza się na każdym poziomie testowania, ale z reguły jest stosowane w testach systemowych. Istnieją standardowe taksonomie, które mają zastosowanie w wielu różnych rodzajach oprogramowania. Uniezależnienie testowania od produktu umożliwia wykorzystanie standardowej wiedzy branżowej do utworzenia testów. Dzięki zgodności z taksonomiami branżowymi metryki występowania defektów można śledzić na poziomie wielu projektów, a nawet organizacji.

Ograniczenia/trudności
Istnieje wiele taksonomii defektów i niektóre z nich są zorientowane na pewne konkretne typy testów, takie jak testy użyteczności. Ważny jest dobór odpowiedniej taksonomii do testowanego oprogramowania, o ile takie taksonomie istnieją. Dla innowacyjnych produktów takie taksonomie mogą nie być dostępne. Niektóre organizacje opracowały własne taksonomie prawdopodobnych lub często spotykanych defektów. Niezależnie od stosowanej taksonomii 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 praktyce kryteria pokrycia w technikach opartych na defektach są słabiej usystematyzowane niż w technikach opartych na specyfikacji — 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 testów, ale że na podstawie rozważanych defektów nie można już opracować w tej technice więcej przydatnych testów.

Typy defektów
Typy wykrywanych defektów zależą zwykle od używanej taksonomii. Jeżeli użyto taksonomii dla interfejsów użytkownika, większość wykrytych defektów będzie prawdopodobnie związana z interfejsem użytkownika, ale przy okazji tych testów mogą zostać wykryte również inne defekty.

3.3.2 Taksonomie defektów

Taksonomie defektów to podzielone na kategorie listy typów defektów. Listy te mogą być bardzo ogólne i służyć jako wytyczne wysokiego poziomu lub mogą być bardzo szczegółowe. Na przykład taksonomia defektów interfejsów użytkownika może zawierać tak ogólne punkty jak funkcjonalność, obsługa błędów, prezentacja graficzna i wydajność. Szczegółowa taksonomia może zawierać listę wszystkich możliwych obiektów interfejsu użytkownika (zwłaszcza dla interfejsów graficznych) i określać rodzaje nieprawidłowej obsługi tych obiektów, na przykład:

- Pole tekstowe
  - Nie są akceptowane poprawne dane
  - Są akceptowane niepoprawne dane
  - Nie jest sprawdzana długość wprowadzanych danych
  - Nie są wykrywane znaki specjalne
  - Komunikaty o błędach dla użytkownika nie są pomocne
  - Użytkownik nie ma możliwości poprawienia błędnych danych
  - Nie są stosowane reguły
- Pole daty
  - Nie są akceptowane poprawne daty
  - Nie są odrzucane niepoprawne daty
  - Nie są weryfikowane zakresy dat
  - Dane o określonej precyzji nie są poprawnie obsługiwane (np. gg:mm:ss)
  - Użytkownik nie ma możliwości poprawienia błędnych danych
  - Nie są stosowane reguły (np. data końcowa musi być późniejsza niż data początkowa)

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Istnieje wiele taksonomii defektów, począwszy od formalnych taksonomii dostępnych w sprzedaży po wyspecjalizowane taksonomie opracowane przez różnego rodzaju organizacje. Również własne wewnętrzne taksonomie firmy mogą posłużyć do ukierunkowanego wykrywania konkretnych defektów często spotykanych w oprogramowaniu tej firmy. Podczas tworzenia nowej taksonomii defektów lub dostosowywania istniejącej należy najpierw zdefiniować cele tej taksonomii. Celem może być na przykład identyfikowanie w interfejsach użytkownika problemów, które stwierdzono w systemach produkcyjnych, lub identyfikowanie problemów z obsługą pól do wprowadzania danych.

Aby utworzyć taksonomię, należy:

1. Określić cel i pożądany poziom szczegółowości taksonomii
2. Wybrać taksonomię, która posłuży jako podstawa nowej taksonomii
3. Zdefiniować wartości i typowe defekty często napotykane w organizacji lub w ogólnych sytuacjach praktycznych

Im bardziej szczegółowa taksonomia, tym więcej czasu potrzeba na jej opracowanie i pielęgnację, ale osiągnie się dzięki temu większą powtarzalność wyników testów. Szczegółowe taksonomie mogą zawierać nadmiarowe informacje, ale umożliwiają podział pracy w zespole testerów bez utraty informacji lub pokrycia.

Wybór odpowiedniej taksonomii otwiera drogę do konstruowania warunków testowych i przypadków testowych. Taksonomia oparta na ryzyku może ułatwić ukierunkowanie testów na określony obszar ryzyka. Taksonomie można również stosować w testach niefunkcjonalnych takich obszarów, jak użyteczność, wydajność itp. Listy systematyczne są udostępniane w różnego rodzaju publikacjach, przez IEEE i w Internecie.

3.4 Techniki oparte na doświadczeniu

Testowanie oparte na doświadczeniu czerpie z umiejętności i intuicji testerów oraz ich doświadczenia w pracy z aplikacjami i technologiami podobnymi do testowanych. Tego rodzaju testy skutecznie wychwytują defekty, ale nie nadają się tak dobrze jak inne techniki do osiągania określonych poziomów pokrycia testami ani tworzenia procedur testowych, które można ponownie wykorzystać. W sytuacjach, gdy brak dokumentacji systemu, na testowanie przeznaczono bardzo niewiele czasu lub zespół testowy jest bardzo doświadczony w pracy z testowanym systemem, testowanie oparte na doświadczeniu może być dobrą alternatywą dla bardziej ustrukturyzowanych metod. Ta metoda może jednak nie być odpowiednia w systemach, dla których jest wymagana szczegółowa dokumentacja testów, ich duża powtarzalność lub precyzyjne oszacowanie pokrycia produktu testami.

Przy zastosowaniu metod dynamicznych i heurystycznych testerzy wykonują zwykle testy oparte na doświadczeniu, a testowanie można bardziej elastycznie dopasować do zachodzących zdarzeń niż w przypadku metod ze ścisłym planowaniem. 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.

Uwaga: poniżej przedstawiono co prawda pewne wskazówki co do szacowania pokrycia produktu testami, jednak w technikach opartych na doświadczeniu nie istnieją formalne kryteria pokrycia.

3.4.1 Zgadywanie błędów

Stosując technikę 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, wówczas zaprojektuje testy polegające na wprowadzaniu różnego rodzaju wartości w polu hasła w celu sprawdzenia, czy rzeczywiście popełniono taki błąd i czy poskutkował on defektem powodującym awarię po uruchomieniu testów.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Zgadywanie błędów jest nie tylko techniką testowania, ale może również być przydatne podczas analizy ryzyka w celu zidentyfikowania potencjalnych stanów awarii. [Myers79]

Obszar zastosowania
Zgadywanie błędów stosuje się przede wszystkim podczas testów integracyjnych i systemowych, można to jednak robić na dowolnym poziomie testowania. Ta technika często jest stosowana razem z innymi technikami i ułatwia poszerzenie zasięgu istniejących przypadków testowych. Zgadywanie błędów może się także okazać skuteczne podczas testowania nowej wersji oprogramowania pod kątem częstych pomyłek i błędów przed rozpoczęciem bardziej rygorystycznego testowania z udokumentowanymi testami. Do ukierunkowania testów mogą być przydatne listy kontrolne i taksonomie.

Ograniczenia/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 testowany. Zgadywanie błędów jest szeroko stosowane, ale często niedokumentowane, może więc być mniej powtarzalne niż inne formy testowania.

Pokrycie
W przypadku wykorzystania taksonomii pokrycie ustala się na podstawie właściwych usterek danych i typów defektów. Jeżeli nie zastosowano żadnej taksonomii, pokrycie jest ograniczone doświadczeniem i wiedzą testera oraz ilością dostępnego czasu. Skuteczność tej techniki bywa różna w zależności od umiejętności testera wynajdowania problematycznych obszarów.

Typy defektów
Zazwyczaj wykrywane są defekty zdefiniowane w wybranej taksonomii lub „odgadnięte” przez Analityka Testów, które mogą pozostać niewykryte przez testy oparte na specyfikacji.

3.4.2 Testowanie w oparciu o listy kontrolne

W technice testowania w oparciu o listy kontrolne doświadczony Analityk Testów korzysta z wysokopoziomowej, ogólnej listy elementów, na jakie należy zwrócić uwagę, jakie należy sprawdzić lub o jakich należy pamiętać, albo z zestawu reguł czy kryteriów, pod kątem jakich trzeba sprawdzić produkt. Listy kontrolne są tworzone w oparciu o standardy, doświadczenie i inne źródła. Przykładem testu opartego na liście kontrolnej jest lista kontrolna standardów interfejsu użytkownika wykorzystywana jako podstawa do testowania aplikacji.

Obszar zastosowania
Testowanie w oparciu o listy kontrolne 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 może być zaznajomiony z testowaniem interfejsów użytkownika, a niekoniecznie z konkretnym testowanym produktem). 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 można wykorzystywać na wszystkich poziomach testowania, a także w testach regresywnych i 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 mimo wykorzystania tej samej listy kontrolnej. Zwiększa to potencjalnie pokrycie, ale czasem cierpi na tym powtarzalność. Listy kontrolne mogą również w

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

nieuzasadniony sposób zwiększać zaufanie co do rzeczywistego poziomu pokrycia, ponieważ przebieg testów zależy od oceny danego testera. Listy kontrolne można budować na podstawie bardziej szczegółowych przypadków testowych lub list i mają one tendencje do rozrastania się z czasem. Wymagana jest ich pielęgnacja, aby zapewnić, że listy kontrolne pokrywają istotne aspekty testowanego oprogramowania.

Pokrycie
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 obejmują awarie wynikające z wprowadzania różnych danych, wykonywania kroków w różnej kolejności lub indywidualnego modyfikowania przebiegu pracy podczas testowania. Zastosowanie list kontrolnych czyni testy bardziej interesującymi, ponieważ podczas ich wykonywania można używać nowych kombinacji danych i procesów.

3.4.3 Testowanie eksploracyjne

Podczas testowania eksploracyjnego tester uczy się jednocześnie produktu i jego defektów, 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, wykonywanie interaktywnie i twórczo. Technika ta nie wymaga obszernej dokumentacji testowanego systemu i to podejście jest często używane 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.

Ograniczenia/trudności
Testowanie eksploracyjne bywa trudne w zarządzaniu i planowaniu czasowym. Pokrycie testami może być wyrywkowe, a powtarzalność niewielka. Jedną z metod zarządzania testami eksploracyjnymi jest zastosowanie kart opisu testów, jakie mają być zrealizowane w danej sesji, oraz ram czasowych 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 i ustalenia, jakie karty opisu będą potrzebne w następnych sesjach. Spotkania podsumowujące słabo się skalują w przypadku dużych zespołów testowych lub dużych projektów.

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 eksploracyjnymi. 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 trzeba przypomnieć sobie kroki, jakie 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 eksploracyjnej (lub innej sesji testowania opartego na doświadczeniu). Przeszukiwanie takich materiałów pod kątem rzeczywistego powodu awarii może być uciążliwe, ale zapewniają one przynajmniej jakąś formę utrwalenia wykonanych kroków.

Pokrycie
W celu określenia zadań, celów i oczekiwanych produktów można zastosować karty opisu testu. Następnie planuje się sesje eksploracyjne tak, aby zrealizować te cele. Karta opisu testu może również wskazywać

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

obszary, w jakich należy skoncentrować testy, elementy należące do zakresu sesji testowania i pozostające poza tym zakresem oraz zasoby potrzebne do wykonania zaplanowanych testów. W ramach sesji można skoncentrować poszukiwania na określonych typach defektów lub innych potencjalnie problemowych obszarach, które nie wymagają sformalizowania testów w postaci skryptów.

Typy defektów
Defekty najczęściej wykrywane za pomocą testów eksploracyjnych to problemy scenariuszowe, które nie zostały wychwycone podczas testów funkcjonalnych 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 bezpieczeństwem.

3.4.4 Wybór najlepszej techniki

Techniki testowania oparte na defektach lub na doświadczeniu wymagają zastosowania wiedzy o defektach oraz doświadczenia testerskiego w celu takiego ukierunkowania testów, aby zwiększyć skuteczność wykrywania defektów. Testy takie obejmują działania od „szybkich testów", w których nie przewiduje się formalnie określonych działań do wykonania przez testera, poprzez wstępnie zaplanowane sesje aż po udokumentowane sesje. Te metody testowania są niemal zawsze przydatne, jednak szczególnej wartości nabierają w następujących sytuacjach:

- brak dokumentów specyfikacji,
- niewystarczająca lub niskiej jakości dokumentacja testowanego systemu,
- niewystarczająca ilość czasu na zaprojektowanie i utworzenie szczegółowych testów,
- testerzy są doświadczeni w danej dziedzinie i/lub technologii,
- dążenie do maksymalizacji pokrycia produktu testami poprzez poszerzenie zakresu testów poza testy udokumentowane,
- analiza awarii w działaniu operacyjnym.

Techniki oparte na defektach lub na doświadczeniu sprawdzają się również w połączeniu z technikami opartymi na specyfikacji, ponieważ uzupełniają braki w pokryciu wynikające z systemowych słabości tych ostatnich. Podobnie jak w przypadku technik opartych na specyfikacji, nie istnieje najlepsza technika dla wszystkich sytuacji. Analityk Testów powinien rozumieć zalety i wady poszczególnych technik i umieć dobrać najlepszą technikę lub zestaw technik do danej sytuacji: typu projektu, harmonogramu, dostępności informacji, umiejętności testera i innych czynników, które mogą mieć wpływ na ten dobór.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

4. Testowanie atrybutów jakościowych oprogramowania — 120 minut

Słowa kluczowe

testowanie dostępności, testowanie dokładności, atrakcyjność, ocena heurystyczna, testowanie współdziałania, łatwość nauki, łatwość obsługi, testowanie dopasowania, SUMI, zrozumiałość, testowanie użyteczności, WAMMI

Cele nauczania dotyczące testowania charakterystyk jakości oprogramowania

4.2 Charakterystyki jakościowe w testowaniu dziedziny biznesowej

TA-4.2.1 (K2) Kandydat potrafi wyjaśnić na przykładach, jakie techniki testowania są odpowiednie do testowania dokładności, dopasowania, współdziałania i zgodności

TA-4.2.2 (K2) Kandydat potrafi wymienić typowe defekty, jakie powinny zostać wykryte podczas testowania dokładności, dopasowania i współdziałania

TA-4.2.3 (K2) Kandydat potrafi wskazać, w jakim momencie cyklu życia oprogramowania powinna być testowana każda z charakterystyk: dokładność, dopasowanie, współdziałanie

TA-4.2.4 (K4) Kandydat potrafi opisać w skrócie podejścia odpowiednie do zweryfikowania i walidacji implementacji wymagań dotyczących dostępności i spełnienia oczekiwań użytkownika w kontekście danego projektu

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

4.1 Wprowadzenie

W poprzednim rozdziale opisano konkretne techniki, jakimi może posłużyć się tester; tematem tego rozdziału będzie zastosowanie tych technik przy ocenie najważniejszych charakterystyk używanych do opisywania jakości aplikacji lub systemów informatycznych.

Przedmiotem niniejszego sylabusa są charakterystyki jakości, które podlegają ocenie przez Analityka Testów. Atrybuty oceniane przez Technicznego Analityka Testów zostały opisane w sylabusie poziomu zaawansowanego dla Technicznych Analityków Testów. Opis charakterystyk jakości produktu bazuje na normie ISO 9126. Mogą w nim być również uwzględnione informacje pochodzące z innych standardów, takich jak ISO 25000 [ISO25000] (który zastąpił normę ISO 9126). Charakterystyki jakości ISO podzielono na charakterystyki (atrybuty) jakości produktu, z których każda może zawierać charakterystyki podrzędne (atrybuty podrzędne). Są one przedstawione w poniższej tabeli wraz ze wskazaniem, które charakterystyki i charakterystyki podrzędne są opisane w sylabusie dla Analityków Testów, a które w sylabusie dla Technicznych Analityków Testów:

Charakterystyka	**Charakterystyka podrzędna**	**Analityk Testów**	**Techniczny Analityk Testów**
Funkcjonalność	Dokładność, dopasowanie, współdziałanie, zgodność	X	
	Bezpieczeństwo		X
Niezawodność	Dojrzałość (odporność), tolerowanie usterek, odtwarzalność, zgodność		X
Użyteczność	Zrozumiałość, łatwość nauki, łatwość obsługi, atrakcyjność, zgodność	X	
Efektywność	Wydajność (w czasie), zużycie zasobów, zgodność		X
Pielęgnowalność	Zdolność do analizy, modyfikowalność, stabilność, testowalność, zgodność		X
Przenaszalność	Zdolność adaptacyjna, instalowalność, koegzystencja, zastępowalność, zgodność		X

Analityk Testów powinien koncentrować się na charakterystykach jakości oprogramowania związanych z funkcjonalnością i użytecznością. Powinien również wykonywać testy dostępności. Dostępność nie jest wymieniona na liście charakterystyk podrzędnych użyteczności, ale często traktuje się ją jako element tego atrybutu. Testowanie pozostałych charakterystyk jakości uważa się zwykle za zadanie Technicznego Analityka Testów. Taki podział pracy jest stosowany w sylabusach ISTQB, jednak w różnych organizacjach może przybierać różne formy.

Dla każdej z charakterystyk jakości wymieniono charakterystykę podrzędną „zgodność”. W pewnych środowiskach, których dotyczą szczególne standardy bezpieczeństwa lub uregulowania prawne, może być wymagane spełnienie określonych standardów lub norm przez każdą charakterystykę jakości (np. zgodność funkcjonalności może oznaczać, że funkcjonalność jest zgodna z pewnym standardem takim jak użycie określonego protokołu komunikacyjnego w celu przesyłania danych do układu scalonego i ich odczytywania). Te standardy mogą być bardzo różne dla poszczególnych branż i dlatego nie będą tutaj omawiane. Jeżeli Analityk Testów pracuje w środowisku, którego dotyczą wymagania zgodności, powinien rozumieć te wymagania i zapewnić, że zarówno testowanie, jak i dokumentacja testów spełnią te wymagania.

Należy zidentyfikować czynniki ryzyka typowe dla wszystkich charakterystyk i charakterystyk podrzędnych jakości omówionych w tej sekcji, aby można było sformułować i udokumentować odpowiednią strategię testowania. Testowanie charakterystyk jakości wymaga szczególnie starannego doboru właściwego momentu w cyklu życia, niezbędnych narzędzi, a także dostępności oprogramowania i dokumentacji do testowania oraz fachowej wiedzy technicznej. Bez odpowiedniej strategii dla każdej z charakterystyk i

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

specyficznych potrzeb związanych z jej testowaniem tester może nie mieć do dyspozycji wystarczająco dużo czasu na odpowiednie zaplanowanie, przygotowanie i wykonanie testów. Część tych testów, np. testowanie użyteczności, może wymagać przydzielenia szczególnych zasobów ludzkich, szczegółowego zaplanowania, udostępnienia specjalnych laboratoriów i konkretnych narzędzi oraz, w większości przypadków, dużej ilości czasu. Czasem testy użyteczności mogą być wykonywane przez osobną grupę ekspertów ds. użyteczności (interfejsów użytkownika).

Testowanie charakterystyk i charakterystyk podrzędnych jakości musi być powiązane z ogólnym harmonogramem testów i muszą być do niego przydzielone wystarczające zasoby. Każda z tych cech ma specyficzne potrzeby, wiąże się ze specyficznymi problemami i może być testowana w różnych punktach cyklu rozwoju oprogramowania; te kwestie są dokładniej omówione w poniższych sekcjach.

Analityk Testów może nie być odpowiedzialny za charakterystyki jakości, które wymagają bardziej technicznego podejścia, powinien jednak zdawać sobie sprawę z ich natury i rozumieć, w jakich obszarach testy różnych charakterystyk mogą się nakładać. Na przykład produkt, który nie zaliczył testów wydajnościowych, prawdopodobnie nie zaliczy również testów użyteczności, jeżeli działa zbyt wolno, żeby użytkownicy mogli z niego efektywnie korzystać. Podobnie produkt, w którym występują problemy ze współdziałaniem pewnych modułów, nie jest prawdopodobnie gotowy do testów przenaszalności, ponieważ problemy na niższym poziomie będzie trudniej wyizolować w zmienionym środowisku.

4.2 Charakterystyki jakościowe w testowaniu dziedziny biznesowej

Głównym obszarem działań Analityka Testów są testy funkcjonalne. Przedmiotem testów funkcjonalnych jest to, „co" robi dany produkt. Podstawą testów funkcjonalnych jest z reguły dokument wymagań lub specyfikacji, konkretna wiedza z danej dziedziny lub wywnioskowana potrzeba. Testy funkcjonalne mają różną formę w zależności od poziomu testów, na jakim są wykonywane, i ewentualnie również w zależności od cyklu rozwoju oprogramowania. Na przykład test funkcjonalny wykonywany podczas testów integracyjnych sprawdza funkcjonalność łączonych poprzez interfejs modułów, które implementują jedną określoną funkcję. Na poziomie testów systemowych testy funkcjonalne obejmują sprawdzenie funkcjonalności całej aplikacji. W systemach złożonych z podsystemów testy funkcjonalne obejmują całość działania zintegrowanych systemów. W środowisku zwinnym testy funkcjonalne są z reguły ograniczone do funkcjonalności udostępnianej w danej iteracji lub w danym przebiegu; testy regresywne w danej iteracji mogą jednak pokrywać całą udostępnioną do danego momentu funkcjonalność.

W testach funkcjonalnych stosuje się szeroki zakres technik testowania (patrz rozdział 3). Testy funkcjonalne może wykonywać przydzielony do tego zadania tester, ekspert z danej dziedziny lub programista (zazwyczaj na poziomie modułu).

Oprócz testów funkcjonalnych omawianych w tej sekcji w zakresie odpowiedzialności Analityka Testów znajdują się jeszcze dwie charakterystyki jakości, które uznaje się za elementy testów niefunkcjonalnych (sprawdzania, „jak" produkt udostępnia funkcjonalność). Te dwa atrybuty niefunkcjonalne to użyteczność i dostępność.

W tej sekcji zostaną omówione następujące charakterystyki jakości:

- funkcjonalne charakterystyki podrzędne jakości:
  - dokładność,
  - dopasowanie,
  - współdziałanie;
- niefunkcjonalne charakterystyki jakości:
  - użyteczność,
  - dostępność.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

4.2.1 Testowanie dokładności

Testowanie dokładności funkcjonalnej obejmuje testowanie zgodności aplikacji z podanymi lub wywnioskowanymi wymaganiami i może obejmować również testowanie dokładności obliczeniowej. W testowaniu dokładności stosuje się wiele z technik testowania opisanych w rozdziale 3 i często używa się specyfikacji lub wcześniejszej wersji systemu jako wyroczni testowej. Testowanie dokładności może mieć miejsce w dowolnej fazie cyklu życia oprogramowania i jest ukierunkowane na wykrywanie niepoprawnej obsługi danych lub sytuacji.

4.2.2 Testowanie dopasowania

Testowanie dopasowania obejmuje ocenę i walidację odpowiedniego dopasowania zestawu funkcji do konkretnych zadań, które te funkcje mają realizować. Takie testy mogą być oparte na przypadkach użycia. Testowanie dopasowania z reguły ma miejsce podczas testów systemowych, ale może też odbywać się na etapie zaawansowanych testów integracyjnych. Defekty wykryte podczas tych testów wskazują, że system nie zaspokoi potrzeb użytkownika w akceptowalny sposób.

4.2.3 Testowanie współdziałania

W ramach testowania współdziałania sprawdza się możliwość wymiany informacji między dwoma lub większą liczbą systemów lub modułów i możliwość późniejszego wykorzystania przez nie tych informacji. Testy muszą pokrywać wszystkie przewidywane środowiska docelowe (w tym różnice w zakresie sprzętu, oprogramowania, oprogramowania warstwy pośredniej, systemów operacyjnych itp.), aby zapewnić prawidłowe działanie 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 systemów oprogramowania wiąże się ze sposobem ich interakcji między sobą. Oprogramowanie o dobrych wskaźnikach współdziałania można zintegrować z wieloma innymi systemami bez wprowadzania w nim większych zmian. Jako miary współdziałania można użyć liczby koniecznych zmian i nakładu pracy potrzebnego na ich realizację.

W testowaniu współdziałania oprogramowania można na przykład koncentrować się na następujących cechach projektowych:

- wykorzystaniu branżowych standardów komunikacji takich jak XML,
- możliwości automatycznego wykrywania potrzeb komunikacyjnych związanych z systemami, z którymi oprogramowanie współdziała, i odpowiednie dostosowanie przesyłanych danych.

Testowanie współdziałania może być szczególnie istotne dla organizacji produkujących oprogramowanie i narzędzia „z półki” (COTS — Commercial Off-The-Shelf Software) oraz w przypadku systemów złożonych z podsystemów.

Tego rodzaju testy wykonuje się podczas testów integracyjnych modułów i testów systemowych, koncentrując się przy tym na interakcjach systemu ze środowiskiem. 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 wykreować warunki, które umożliwią przetestowanie różnych interakcji. Na przykład, jeżeli między dwoma systemami ma zachodzić wymiana danych, Analityk Testów musi umieć 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 umieć przeanalizować te dokumenty w celu ustalenia punktów wymiany informacji między systemami oraz między systemem a jego środowiskiem, aby zapewnić przetestowanie ich wszystkich, i być przygotowany

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

na otrzymanie takiego zadania. W testowaniu współdziałania stosuje się takie techniki jak tablice decyzyjne, diagramy przejść pomiędzy stanami, przypadki użycia i testowanie kombinatoryjne. Wykrywane defekty z reguły dotyczą niepoprawnej wymiany danych między komponentami wchodzącymi ze sobą w interakcje.

4.2.4 Testowanie użyteczności

Należy rozumieć, z jakich powodów użytkownicy mogą mieć problemy z korzystaniem z systemu. Aby to osiągnąć, należy przede wszystkim uwzględnić, że pojęcie „użytkownik" może odnosić się do wielu różnych typów ludzi, od ekspertów IT poprzez dzieci po osoby niepełnosprawne.

Niektóre instytucje krajowe (np. Królewski Narodowy Instytut Niewidomych w Wielkiej Brytanii) zalecają, aby strony WWW były dostępne dla osób niepełnosprawnych, niewidomych, niedowidzących, niepełnosprawnych ruchowo, niesłyszących i niepełnosprawnych intelektualnie. Sprawdzenie, czy tacy użytkownicy mogą bez przeszkód korzystać z aplikacji i stron WWW może poprawić również użyteczność tych produktów dla innych użytkowników. Więcej informacji o dostępności można znaleźć dalej w treści niniejszego dokumentu.

Podczas testowania użyteczności sprawdza się, czy użytkownicy mogą łatwo korzystać lub nauczyć się korzystać z systemu w celu osiągnięcia konkretnego celu w konkretnym kontekście. Testowanie użyteczności ma mierzyć następujące cechy:

- skuteczność — możliwość osiągnięcia przez użytkowników przy użyciu produktu określonych celów z zachowaniem dokładności i kompletności w określonym kontekście użycia,
- efektywność — możliwość uzyskania przez użytkowników przy użyciu produktu określonej efektywności w określonym kontekście użycia przy odpowiednich nakładach zużytych zasobów,
- satysfakcję — możliwość zadowolenia użytkowników z produktu w określonym kontekście użycia.

Atrybuty, które można mierzyć, to m. in.:

- zrozumiałość — atrybuty oprogramowania, które wpływają na nakład pracy wymagany do zrozumienia przez użytkownika koncepcji logicznej produktu i jej zastosowania do danego zadania,
- łatwość nauki — atrybuty oprogramowania, które wpływają na nakład pracy wymagany do opanowania przez użytkownika korzystania z aplikacji,
- łatwość obsługi — atrybuty oprogramowania, które wpływają na nakład pracy wymagany do skutecznego i efektywnego wykonywania zadań przez użytkownika,
- atrakcyjność — możliwość polubienia oprogramowania przez użytkownika.

Testy użyteczności odbywają się zwykle w dwóch krokach:

- kształtujące testy użyteczności — testy wykonywane iteracyjnie w fazach projektowania i prototypowania w celu ukierunkowania („ukształtowania") projektu poprzez identyfikowanie defektów projektu użyteczności,
- podsumowujące testy użyteczności — testy wykonywane po implementacji w celu pomiaru użyteczności i zidentyfikowania problemów z ukończonym komponentem lub systemem.

Zestaw umiejętności testera użyteczności powinien obejmować wiedzę lub doświadczenie w następujących dziedzinach:

- socjologia,
- psychologia,
- zgodność ze standardami krajowymi (w tym ze standardami dostępności),
- ergonomia.

4.2.4.1 Wykonywanie testów użyteczności

Walidacja implementacji produktu powinna się odbywać w warunkach jak najbardziej zbliżonych do rzeczywistych warunków użytkowania systemu. Może to oznaczać konieczność urządzenia laboratorium użyteczności z kamerami, imitacjami biur, panelami przeglądowymi, użytkownikami itp., tak aby programiści mogli zaobserwować efekty rzeczywistego korzystania z systemu przez rzeczywistych użytkowników.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Formalne testy użyteczności często wymagają pewnego przygotowania „użytkowników” (rzeczywistych użytkowników lub ich przedstawicieli) poprzez udostępnienie szczegółowych skryptów lub instrukcji, jakimi mają się kierować podczas testowania. Inne, nieudokumentowane testy umożliwiają użytkownikom eksperymentowanie z oprogramowaniem, a obserwatorom ustalenie, jak łatwo (lub trudno) jest użytkownikom zorientować się, jak wykonać określone zadania.

Wiele testów użyteczności może zostać wykonanych przez Analityka Testów w ramach innych testów, na przykład podczas systemowych testów funkcjonalnych. Do uzyskania spójnego podejścia do wykrywania i zgłaszania defektów użyteczności na wszystkich etapach cyklu życia oprogramowania mogą być przydatne wytyczne użyteczności. Bez wytycznych użyteczności może być trudno określić, co jest nie do zaakceptowania z punktu widzenia użyteczności. Na przykład czy jest nierozsądne rozwiązanie, w którym użytkownik będzie potrzebować 10 kliknięć, aby zalogować się do aplikacji? Bez konkretnych wytycznych Analityk Testów może znaleźć się w trudnej sytuacji, kiedy będzie musiał bronić zgłoszeń defektów, które programiści będą chcieli zamknąć z uzasadnieniem, że oprogramowania działa „zgodnie z projektem”. Bardzo ważne jest zdefiniowanie weryfikowalnej specyfikacji użyteczności w ramach wymagań oraz posiadanie zestawu wytycznych użyteczności, stosowanych do wszystkich projektów danego rodzaju. W wytycznych powinny być uwzględnione takie elementy, jak dostępność instrukcji, czytelność komunikatów zachęty, liczba kliknięć konieczna do wykonania danej czynności, komunikaty o błędach, wskaźniki przetwarzania (wskaźniki informujące użytkownika, że system przetwarza dane i nie przyjmuje w danej chwili nowo wprowadzanych danych), układ elementów na ekranie, wykorzystanie kolorów i dźwięków oraz inne czynniki, które wpływają na wrażenia użytkownika.

4.2.4.2 Specyfikacja testów użyteczności

Najważniejsze techniki testowania użyteczności to:

- inspekcja, ocena i przegląd,
- dynamiczne interakcje z prototypami,
- weryfikacja i walidacja implementacji,
- ankiety i kwestionariusze.

Inspekcja, ocena i przegląd

Inspekcje lub przeglądy specyfikacji wymagań i dokumentacji projektów pod kątem użyteczności, które zwiększają zaangażowanie użytkownika w projekt, mogą mieć walory ekonomiczne dzięki wczesnemu wykrywaniu problemów. Można wykorzystać ocenę heurystyczną (systematyczną inspekcję projektu interfejsu użytkownika pod kątem użyteczności) w celu zidentyfikowania problemów z użytecznością podczas projektowania, aby można im było zaradzić w ramach iteracyjnego procesu projektowania. Wymaga to zaangażowania niewielkiego zespołu oceniającego, który zbada interfejs i oceni jego zgodność z uznawanymi regułami użyteczności („heurystyką”). 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.

Dynamiczne interakcje z prototypami

Na etapie projektowania prototypów Analityk Testów powinien używać prototypów i wspierać projektantów w ich ulepszaniu poprzez dostarczanie informacji zwrotnych z perspektywy użytkownika. Umożliwia to udoskonalenie prototypów i uzyskanie przez użytkownika realistycznego wyobrażenia o wyglądzie gotowego produktu i interakcjach z nim.

Weryfikacja i walidacja implementacji

Jeżeli w wymaganiach określono charakterystyki użyteczności oprogramowania (np. liczba kliknięć niezbędnych do osiągnięcia jakiegoś celu), należy utworzyć przypadki testowe, które będą weryfikować uwzględnienie tych charakterystyk w implementacji tego oprogramowania.

W celu walidacji implementacji można skorzystać z testów zdefiniowanych dla potrzeb systemowych testów funkcjonalnych jako ze scenariuszy testów użyteczności. Te scenariusze testowe mierzą konkretne charakterystyki użyteczności, takie jak łatwość nauki czy obsługi, zamiast rezultatów funkcjonalnych.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

Scenariusze testowe użyteczności mogą być opracowane zgodnie z określoną składnią i semantyką testów. Składnia to struktura lub „gramatyka" interfejsu (np. typ wartości, jakie można wprowadzić w określonym polu), a semantyka opisuje znaczenie i cel interfejsu (np. uzasadnione i treściwe komunikaty systemowe i dotyczące wyników przetwarzania, jakie są udostępniane użytkownikowi).

W testowaniu użyteczności stosuje się czasem techniki czarnoskrzynkowe (np. te opisane w sekcji 3.2), zwłaszcza przypadki użycia, które można zdefiniować w postaci tekstowej lub w języku UML (Unified Modeling Language).

W scenariuszach testów użyteczności należy również uwzględnić instrukcje dla użytkownika, przydział czasu na wywiady przed testami i po nich (w celu udzielenia instrukcji i zebrania informacji zwrotnych) oraz ustalony protokół prowadzenia sesji testowych. Protokół obejmuje opis wykorzystywanego sposobu wykonywania testów, przydzielonych czasów, metod notowania i rejestrowania podczas sesji oraz metod prowadzenia wywiadów i ankiet.

Ankiety i kwestionariusze
Do gromadzenia obserwacji i informacji zwrotnych związanych z zachowaniem użytkowników systemu można zastosować techniki ankiet i kwestionariuszy. Standardowe, ogólnodostępne ankiety takie jak SUMI (Software Usability Measurement Inventory) i WAMMI (Website Analysis and MeasureMent Inventory) umożliwiają porównywanie rezultatów ze zgromadzonymi w bazie danych pomiarami dokonanymi w innych projektach. Ponadto ankieta SUMI uwzględnia konkretne metryki użyteczności, może zatem służyć jako źródło zbioru kryteriów ukończenia/akceptacji.

4.2.5 Testowanie dostępności

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, oraz uregulowania prawne, takie jak Ustawa o dyskryminacji ze względu na niepełnosprawność (Wielka Brytania, Australia) czy Sekcja 508 (USA). Podobnie jak w przypadku użyteczności, dostępnością należy interesować 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, gdy oprogramowanie nie spełnia właściwych uregulowań prawnych lub standardów.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

5. Przeglądy — 165 minut

Słowa kluczowe
brak

Cele nauczania dotyczące przeglądów

5.1 Wprowadzenie
TA-5.1.1 (K2) Kandydat potrafi wyjaśnić, dlaczego przygotowanie do przeglądu jest istotne w przypadku Analityka Testów

5.2 Korzystanie z list kontrolnych podczas przeglądów
TA-5.2.1 (K4) Kandydat potrafi przeanalizować przypadek użycia lub interfejs użytkownika i zidentyfikować problemy zgodnie z informacjami z list kontrolnych podanych w sylabusie
TA-5.2.2 (K4) Kandydat potrafi przeanalizować specyfikację wymagań lub historyjkę użytkownika i zidentyfikować problemy zgodnie z informacjami z list kontrolnych podanych w sylabusie

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

5.1 Wprowadzenie

Skuteczne przeprowadzenie przeglądu wymaga przygotowania, aktywnego uczestnictwa i zastosowania wyników. Analityk Testów musi aktywnie uczestniczyć w procesie przeglądu i korzystać ze swojej specyficznej perspektywy. Powinien zostać przeszkolony w zakresie przeglądów formalnych tak, aby rozumiał swoją rolę w dowolnym procesie przeglądu. Wszyscy uczestnicy przeglądu muszą dążyć do osiągnięcia korzyści płynących z dobrze przeprowadzonego przeglądu. Prawidłowo dokonany przegląd może być największym i najbardziej efektywnym ekonomicznie elementem ogólnej jakości dostarczanego produktu.

Niezależnie od typu dokonywanego przeglądu Analityk Testów musi przewidzieć odpowiednio dużo czasu na przygotowanie do niego. Przygotowanie obejmuje czas potrzebny na przejrzenie produktu, na sprawdzenie powiązanych dokumentów pomocniczych pod kątem spójności oraz na ustalenie, czego może brakować w danym produkcie. Gdy na przygotowanie nie zostanie przewidziana odpowiednia ilość czasu, Analityk Testów może zostać zmuszony do ograniczenia się do analizy tego, co już jest zawarte w dokumencie, zamiast wnieść swój wkład w efektywny przegląd, który maksymalnie wykorzystuje czas poświęcony przez zespół i skutkuje jak najlepszymi informacjami zwrotnymi. Dobrze przeprowadzony przegląd obejmuje zrozumienie tego, co zostało napisane, ustalenie, czego brakuje i sprawdzenie, czy dany produkt jest spójny z innymi produktami (już opracowanymi albo dopiero przygotowywanymi). Na przykład podczas przeglądu planu testów na poziomie integracji Analityk Testów musi również uwzględnić integrowane elementy. Jakie warunki muszą być spełnione, aby były one gotowe do integracji? Czy istnieją jakieś zależności, które powinny zostać udokumentowane? Czy są dostępne dane do testowania punktów integracji? Przegląd nie ogranicza się do produktu podlegającego przeglądowi, ale należy w nim również uwzględnić powiązania tego produktu z innymi produktami w systemie.

Łatwo może się zdarzyć, że autor produktu podlegającego przeglądowi poczuje się krytykowany. Wszystkie komentarze Analityka Testów w ramach przeglądu powinny być wygłaszane z perspektywy współpracy z autorem w celu uzyskania jak najlepszej jakości produktu. Umożliwia to konstruktywne formułowanie komentarzy, które będą się odnosić do produktów, a nie do autora. Na przykład w przypadku niejednoznacznego stwierdzenia lepiej powiedzieć „Nie rozumiem, co należy przetestować, żeby sprawdzić, że to wymaganie zostało poprawnie zaimplementowane. Czy możesz mi pomóc to zrozumieć?” niż „To wymaganie jest niejednoznaczne i nikt go nie zrozumie”. Zadaniem Analityka Testów w ramach przeglądu jest zapewnienie, że informacje zawarte w produkcie wystarczą do wykonania testów. Jeżeli informacji brakuje, są one niejasne lub niewystarczająco szczegółowe, wówczas stanowią prawdopodobnie defekt, który powinien zostać usunięty przez autora. Konstruktywne, a nie krytyczne podejście ułatwia adresatowi przyjęcie komentarzy i zwiększa produktywność spotkania.

5.2 Korzystanie z list kontrolnych podczas przeglądów

Podczas przeglądów używa się list kontrolnych, aby przypomnieć uczestnikom o sprawdzeniu w ramach przeglądu konkretnych elementów. Pomagają one również w wyeliminowaniu czynnika osobistego w przeglądzie: „używamy tej samej listy we wszystkich przeglądach, nie tylko do twojego produktu”. Listy kontrolne mogą być ogólne, do zastosowania we wszelkiego rodzaju przeglądach, lub skoncentrowane na określonych atrybutach jakości, obszarach albo typach dokumentów. Na przykład ogólna lista kontrolna może służyć do weryfikacji ogólnych właściwości dokumentu, jak niepowtarzalny 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 pojęć „będzie” i „powinien”, testowalności każdego z wymagań itp. Również sam format wymagań może dyktować typ używanej listy kontrolnej. Do dokumentacji wymagań w postaci opisu słownego należy zastosować inne kryteria przeglądu niż do dokumentacji opartej na diagramach.

Listy kontrolne mogą być również profilowane pod kątem kompetencji programisty/architekta albo kompetencji testera. W przypadku Analityka Testów odpowiednia będzie lista kontrolna zorientowana na

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

kompetencje testerskie. Przykłady punktów, jakie mogą zawierać tego rodzaju listy kontrolne, podano poniżej.

Listy kontrolne używane do przeglądów wymagań, przypadków użycia i historyjek użytkownika zwykle zorientowane są na inne kryteria niż te używane do przeglądów kodu czy architektury. Na liście kontrolnej do przeglądów wymagań mogą się znaleźć na przykład następujące punkty:

- testowalność każdego z wymagań,
- kryteria akceptacji dla każdego z wymagań,
- dostępność struktury wywołań przypadków użycia, o ile ma ona zastosowanie,
- jednoznaczna identyfikacja każdego wymagania / przypadku użycia lub każdej historyjki użytkownika,
- kontrola wersji każdego wymagania / przypadku użycia lub każdej historyjki użytkownika,
- możliwość prześledzenia wynikania każdego wymagania z wymagań biznesowych lub marketingowych,
- możliwość prześledzenia związków między wymaganiami a przypadkami użycia.

Powyższe punkty to jedynie przykłady. Należy pamiętać, że jeżeli wymaganie nie jest testowalne, czyli jest zdefiniowane w taki sposób, że Analityk Testów nie może ustalić sposobu jego przetestowania, to w takim wymaganiu tkwi defekt. Na przykład wymaganie „Oprogramowanie powinno być bardzo wygodne w obsłudze" nie jest testowalne. Jak Analityk Testów ma ustalić, czy oprogramowanie jest wygodne w obsłudze, a tym bardziej „bardzo wygodne w obsłudze"? Gdyby zamiast tego wymaganie zawierało stwierdzenie „Oprogramowanie musi być zgodne ze standardami użyteczności określonymi w dokumencie standardów użyteczności" i gdyby taki dokument standardów użyteczności rzeczywiście istniał, wówczas to wymaganie byłoby testowalne. Jest to też wymaganie bardzo ogólne, 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 albo 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 do przeglądów przypadków użycia może zawierać następujące pytania:

- Czy ścieżka główna (scenariusz główny) jest jasno zdefiniowana?
- Czy zostały zidentyfikowane wszystkie ścieżki (scenariusze) alternatywne wraz z obsługą błędów?
- Czy zostały zdefiniowane komunikaty interfejsu użytkownika?
- Czy istnieje tylko jedna ścieżka główna (tak powinno być – jeżeli jest inaczej, mamy do czynienia z więcej niż jednym przypadkiem użycia)?
- Czy każda ze ścieżek jest testowalna?

Prosta lista kontrolna użyteczności interfejsu użytkownika może zawierać następujące punkty:

- Czy zdefiniowano każde z pól i jego funkcję?
- Czy zdefiniowano wszystkie komunikaty o błędach?
- Czy zdefiniowano wszystkie zapytania kierowane do użytkownika i czy są one spójne?
- Czy zdefiniowano kolejność przechodzenia między polami za pomocą klawisza tabulacji?
- Czy istnieją skróty klawiszowe dla działań wykonywanych za pomocą myszy?
- Czy zdefiniowano skróty klawiszowe dla operacji użytkownika (np. wycinania i wklejania)?
- Czy istnieją zależności między polami (np. jakaś data musi być późniejsza od innej daty)?
- Czy opracowano układ ekranu?
- Czy układ ekranu jest zgodny z podanymi wymaganiami?
- Czy podczas przetwarzania danych w systemie zostaje wyświetlony użytkownikowi jakiś wskaźnik tego przetwarzania?

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

- Czy dany ekran spełnia wymaganie jak najmniejszej liczby kliknięć (o ile zostało ono zdefiniowane)?
- Czy przepływ nawigacji jest logiczny z punktu widzenia użytkownika opisanego w przypadku użycia?
- Czy dany ekran spełnia ewentualne wymagania co do łatwości nauki?
- Czy użytkownik ma do dyspozycji tekst pomocy?
- Czy użytkownik ma do dyspozycji dymki wyświetlane po wskazaniu elementów kursorem myszy?
- Czy użytkownik uzna dany element za „atrakcyjny" (ocena subiektywna)?
- Czy użyto kolorów zgodnie z konwencją stosowaną w innych aplikacjach i ze standardami organizacji?
- Czy efekty dźwiękowe są prawidłowo używane i czy można je konfigurować?
- Czy ekran spełnia wymagania związane z lokalizacją?
- Czy użytkownik jest w stanie określić, co ma zrobić (zrozumiałość) (ocena subiektywna)?
- Czy użytkownik zapamięta, co ma zrobić (łatwość nauki) (ocena subiektywna)?

W projektach zwinnych 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 i jej zakres jest uzależniony od czasu potrzebnego na zaimplementowanie danej funkcjonalności. Lista kontrolna do przeglądów historyjek użytkownika może zawierać następujące pytania:

- Czy historyjka jest odpowiednia dla docelowej iteracji / docelowego przebiegu?
- Czy zostały zdefiniowane kryteria akceptacji i czy są one testowalne?
- Czy funkcjonalność została jasno zdefiniowana?
- Czy istnieją zależności między tą historyjką a innymi historyjkami?
- Czy historyjce przypisano priorytet?
- Czy historyjka opisuje jedną funkcjonalność?

Oczywiście 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.

Listę kontrolną można modyfikować w zależności od następujących czynników:

- organizacja (np. w celu uwzględnienia polityk, standardów i konwencji firmowych),
- konkretny projekt lub konkretne prace rozwojowe (np. specyfika projektu, standardy techniczne, ryzyka),
- obiekt poddawany przeglądowi (np. przeglądy kodu mogą być dostosowywane do specyfiki konkretnych języków programowania).

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

Więcej informacji o przeglądach i inspekcjach można znaleźć w [Gilb93] i [Wiegers03].

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

6. Zarządzanie defektami — 120 minut

Słowa kluczowe

Taksonomia defektów, powstrzymanie fazowe, analiza przyczyny podstawowej

Cele nauczania dotyczące zarządzania defektami

6.2 Kiedy można wykryć defekt?

TA-6.2.1 (K2) Kandydat potrafi wyjaśnić, w jaki sposób powstrzymanie fazowe umożliwia obniżenie kosztów

6.3 Pola zgłoszenia defektu

TA-6.3.1 (K2) Kandydat potrafi wyjaśnić, jakie informacje mogą być potrzebne przy dokumentowaniu defektu niefunkcjonalnego

6.4 Klasyfikacja defektów

TA-6.4.1 (K4) Kandydat potrafi zidentyfikować, zgromadzić i wypisać informacje dotyczące klasyfikacji podanego defektu

6.5 Analiza przyczyny podstawowej

TA-6.5.1 (K2) Kandydat potrafi wyjaśnić cel analizy przyczyny podstawowej

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

6.1 Wprowadzenie

Analityk Testów ocenia zachowanie systemu pod kątem potrzeb biznesowych i potrzeb użytkowników, np. sprawdzając, czy użytkownik wiedziałby, co zrobić, gdy zostanie wyświetlony dany komunikat albo gdy system wykona dane działanie. Porównanie rezultatu rzeczywistego z oczekiwanym umożliwia Analitykowi Testów określenie, czy system działa prawidłowo. Anomalie (nazywane też incydentami) to nieoczekiwane zdarzenia, które wymagają dokładniejszego zbadania. Anomalią może być awaria spowodowana defektem. Napotkanie anomalii może, ale nie musi, skutkować utworzeniem zgłoszenia defektu. Defekt to rzeczywisty problem, który powinien zostać rozwiązany.

6.2 Kiedy można wykryć defekt?

Defekt można wykryć poprzez analizę statyczną, a jego objawy, czyli awarię, poprzez testowanie dynamiczne. Każda faza cyklu rozwoju oprogramowania powinna uwzględniać metody wykrywania i eliminowania potencjalnych awarii. Na przykład w fazie wytwarzania oprogramowania w celu wykrywania defektów należy stosować przeglądy kodu i projektów. Podczas testowania dynamicznego do wykrywania awarii służą przypadki testowe.

Im wcześniej defekt zostanie wykryty i usunięty, tym niższy jest całkowity koszt jakości systemu. Na przykład analiza statyczna umożliwia wykrywanie defektów jeszcze zanim testowanie dynamiczne stanie się możliwe. Jest to jeden z powodów, dla których analiza statyczna jest opłacalnym sposobem na wytwarzanie oprogramowania wysokiej jakości.

System śledzenia defektów powinien umożliwiać Analitykowi Testów zarejestrowanie fazy cyklu rozwoju oprogramowania, w której defekt został wprowadzony i fazy, w której został wykryty. Jeżeli te dwie fazy są takie same, udało się osiągnąć doskonałe powstrzymanie fazowe. Oznacza to, że defekt został wprowadzony i wykryty w tej samej fazie i nie „wyciekł" do kolejnego etapu. Przykładem takiej sytuacji może być niepoprawne wymaganie, które zostało znalezione i poprawione w ramach przeglądu wymagań. Po pierwsze, jest to efektywne wykorzystanie przeglądu wymagań, a po drugie zapobiega powstaniu dodatkowego nakładu pracy związanego z tym defektem, a przez to powstaniu dodatkowych kosztów dla organizacji. Jeżeli niepoprawne wymaganie „umknie" w przeglądzie wymagań i zostanie zaimplementowane przez programistę, przetestowane przez Analityka Testów i wykryte dopiero przez użytkownika podczas testów akceptacyjnych, cała włożona w nie praca idzie na marne (a użytkownik może stracić zaufanie do poprawności systemu).

Powstrzymanie fazowe to efektywny sposób na redukcję kosztów defektów.

6.3 Pola zgłoszenia defektu

Pola (parametry) zgłoszenia defektu umożliwiają udostępnienie ilości informacji wystarczającej do podjęcia działań związanych z tym defektem. Zgłoszenie defektu umożliwia podjęcie działań związanych z defektem, jeżeli jest:

- kompletne — zawiera wszystkie niezbędne informacje,
- zwięzłe — nie zawiera niepotrzebnych informacji,
- dokładne — informacje zawarte w zgłoszeniu są dokładne, rezultaty oczekiwany i rzeczywisty sprecyzowane i podano odpowiednie kroki umożliwiające odtworzenie problemu,
- obiektywne — zgłoszenie jest napisane profesjonalnym językiem i jest rzeczowe.

Informacje ujęte w zgłoszeniu defektu powinny być umieszczone w odpowiednich polach danych. Im lepiej zdefiniowane pola, tym łatwiej zgłaszać poszczególne defekty oraz generować raporty o trendach i inne podsumowania. Jeżeli dane pole może przyjmować tylko określoną liczbę wartości, czas potrzebny na zgłoszenie defektu może być skrócony przez użycie listy rozwijanej z dostępnymi wartościami. Listy rozwijane sprawdzają się tylko wtedy, gdy liczba opcji jest niewielka i użytkownik nie musi przewijać długiej listy w poszukiwaniu właściwej wartości. W różnych rodzajach zgłoszeń są wymagane różne informacje i narzędzie do zarządzania defektami powinno elastycznie podpowiadać właściwe pola zależnie od typu

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

defektu. Dane należy rejestrować w oddzielnych polach, najlepiej z mechanizmem ich walidacji, tak aby nie zdarzały się błędy przy wprowadzaniu danych, a tworzenie zgłoszeń przebiegało efektywnie.

Zgłoszenia defektów tworzy się w związku z awariami napotkanymi podczas testów funkcjonalnych i niefunkcjonalnych. Informacje zawarte w zgłoszeniu defektu powinny zawsze umożliwiać jasną identyfikację scenariusza, za pomocą którego wykryto problem, i zawierać odpowiednie kroki i dane niezbędne do odtworzenia tego scenariusza oraz rezultaty oczekiwane i rzeczywiste. W zgłoszeniach defektów niefunkcjonalnych mogą być niezbędne dodatkowe informacje o środowisku, innych parametrach wydajności (np. wielkość obciążenia), kolejności wykonywania kroków i oczekiwanych rezultatach. Przy dokumentowaniu awarii użyteczności należy określić, jakiego działania oprogramowania oczekiwał użytkownik. Na przykład jeżeli standardem użyteczności jest wykonanie czynności za pomocą mniej niż czterech kliknięć, zgłoszenie defektu powinno zawierać informację, ile kliknięć było koniecznych w odniesieniu do założonego standardu. W sytuacjach, gdy standardy nie są dostępne, a wymagania nie obejmowały niefunkcjonalnych aspektów jakości oprogramowania, tester może posłużyć się próbą „zdrowego rozsądku", aby ustalić, czy użyteczność jest nie do zaakceptowania. W takim przypadku w zgłoszeniu defektu należy jasno określić „zdroworozsądkowe" oczekiwania. Wymagania niefunkcjonalne czasem nie są ujęte w dokumentacji wymagań, więc dokumentowanie awarii niefunkcjonalnych w zakresie działania „oczekiwanego" w zderzeniu z „rzeczywistym" jest trudniejsze.

Poza typowym celem zgłoszenia defektu, czyli uzyskaniem poprawki problemu, informacje o defekcie muszą umożliwiać jego dokładną klasyfikację, analizę ryzyka i udoskonalenia procesu.

6.4 Klasyfikacja defektów

Zgłoszenie defektu może być w cyklu życia klasyfikowane według kilku różnych kryteriów. Właściwe zaklasyfikowanie defektu stanowi integralną część poprawnego zgłaszania defektów. Klasyfikacja stanowi podstawę do grupowania defektów, oceny efektywności testowania i cyklu wytwarzania oprogramowania oraz do obserwowania interesujących trendów.

Typowe informacje klasyfikacyjne nowo zidentyfikowanych defektów to m. in.:

- działanie projektowe, w wyniku którego wykryto defekt — np. przegląd, audyt, inspekcja, tworzenie kodu, testowanie,
- faza projektu, w której wprowadzono defekt (o ile jest znana) — np. gromadzenie wymagań, projektowanie, projektowanie niskopoziomowe, tworzenie kodu,
- faza projektu, w której wykryto defekt — np. gromadzenie wymagań, projektowanie, projektowanie niskopoziomowe, tworzenie kodu, przegląd kodu, testy modułowe, testy integracyjne, testy systemowe, testy akceptacyjne,
- podejrzenie co do przyczyny defektu — np. wymagania, projekt, interfejs, kod, dane,
- powtarzalność — np. jednorazowy, nieregularny, powtarzalny,
- symptom — np. nieprawidłowe zakończenie działania, zawieszenie programu, błąd interfejsu użytkownika, błąd systemu, problem wydajnościowy.

Po zbadaniu problemu można go sklasyfikować według dodatkowych kryteriów:

- przyczyna podstawowa — popełniony błąd, wskutek którego powstał defekt, np. proces, błąd przy tworzeniu kodu, błąd użytkownika, błąd w teście, problem z konfiguracją, problem z danymi, oprogramowanie innych firm, zewnętrzny problem z oprogramowaniem, problem z dokumentacją,
- źródło — produkt, w którym popełniono błąd, np. wymagania, projekt, projekt niskopoziomowy, architektura, projekt bazy danych, dokumentacja dla użytkowników, dokumentacja testów,
- typ — np. problem logiczny, problem obliczeniowy, problem z rozłożeniem działań w czasie, obsługa danych, udoskonalenie.

Po usunięciu defektu (lub jego odłożeniu do usunięcia w przyszłości albo ustaleniu, że nie da się go potwierdzić) mogą się pojawić dalsze informacje służące za podstawę klasyfikacji, np.:

- rozwiązanie — np. zmiana w kodzie, zmiana w dokumentacji, odłożony, nie stanowi problemu, duplikat,

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

- działanie naprawcze — np. przegląd wymagań, przegląd kodu, test modułowy, udokumentowanie konfiguracji, przygotowanie danych, nie dokonano zmian.

Oprócz tych kategorii defekty często klasyfikuje się na podstawie ich wagi i priorytetu. Ponadto zależnie od projektu może mieć sens klasyfikacja oparta na wpływie na bezpieczeństwo kluczowych działań organizacji, wpływie na harmonogram projektu, kosztach projektowych, ryzyku projektowym oraz wpływie na jakość projektu. Tego rodzaju klasyfikacje bywają używane w umowach regulujących czasy dostarczania poprawek.

Ostatnią metodą klasyfikacji jest końcowe rozwiązanie. Defekty często grupuje się w oparciu o rozwiązanie, np. naprawiony/sprawdzony, zamknięty/nie stanowi problemu, odłożony, otwarty/nierozwiązany. Tej klasyfikacji używa się w całym okresie trwania projektu przy śledzeniu defektów na przestrzeni ich cyklu życia.

Wartości używane przez poszczególne organizacje do klasyfikowania są często dopasowywane do potrzeb tych organizacji. Powyższe przykłady to tylko niektóre rodzaje powszechnie stosowane w branży. Aby wartości klasyfikacyjne były przydatne, należy ich używać konsekwentnie. Zbyt wiele pól klasyfikacyjnych wymaga dodatkowego czasu podczas otwierania i przetwarzania zgłoszenia defektu, jest więc istotne rozważenie przydatności gromadzonych danych w odniesieniu do narastających kosztów przetwarzania kolejnych defektów. Możliwość dostosowywania wartości klasyfikacyjnych w narzędziu do zarządzania defektami często stanowi istotny argument przy podejmowaniu decyzji o wyborze takiego narzędzia.

6.5 Analiza przyczyny podstawowej

Celem analizy przyczyny podstawowej jest ustalenie, co spowodowało wystąpienie defektu i udostępnienie danych wspierających takie zmiany procesów, które umożliwią wyeliminowanie przyczyn podstawowych dotyczących większej liczby defektów. Analizę przyczyny podstawowej wykonuje z reguły osoba, która bada problem i albo go rozwiązuje, albo ustala, że nie można lub nie należy go rozwiązać. Zwykle jest to programista. Analityk Testów z reguły wstępnie określa przyczynę podstawową, podając swoje uzasadnione podejrzenia co do prawdopodobnej przyczyny problemu. Przy potwierdzaniu poprawki Analityk Testów weryfikuje informację o przyczynie podstawowej podaną przez programistę. Przy ustalaniu przyczyny podstawowej ustala się zwykle lub potwierdza fazę, w której wprowadzono defekt.

Typowe przyczyny podstawowe to m. in.:
- niejasne wymaganie,
- brak wymagania,
- błędne wymaganie,
- niepoprawna implementacja projektu,
- niepoprawna implementacja interfejsu,
- błąd logiki w kodzie,
- błąd obliczeniowy,
- błąd sprzętowy,
- błąd interfejsu,
- niepoprawne dane.

Informacje o przyczynie podstawowej są agregowane w celu ustalenia typowych problemów, które powodują powstawanie defektów. Na przykład jeżeli duża liczba defektów jest spowodowana niejasnymi wymaganiami, sensowne będzie przeznaczenie większego nakładu pracy na skuteczne przeglądy wymagań. Podobnie, jeżeli w różnych zespołach programistów problem stanowi implementacja interfejsów między modułami, mogą się okazać potrzebne wspólne sesje projektowania.

Wykorzystanie informacji o przyczynach podstawowych do udoskonalania procesów ułatwia organizacji monitorowanie korzyści płynących z efektywnych zmian procesów i obliczanie kosztów defektów związanych z poszczególnymi przyczynami podstawowymi. Dzięki temu może być łatwiej zdobyć finansowanie zmian procesów, które wymagają zakupu dodatkowych narzędzi i wyposażenia lub

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

modyfikacji harmonogramów. Analiza przyczyny podstawowej jest omówiona szerzej w sylabusie ISTQB poziomu eksperckiego „Improving the Test Process” [ISTQB_EL_ITP].

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

7. Narzędzia testowe — 45 minut

Słowa kluczowe

testowanie oparte o słowa kluczowe, narzędzie do przygotowywania danych testowych, narzędzie do projektowania testów, narzędzie do automatyzacji testów

Cele nauczania dotyczące narzędzi testowych

7.2 Narzędzia testowe i automatyzacja testów

TA-7.2.1 (K2) Kandydat potrafi wyjaśnić korzyści związane z zastosowaniem narzędzi do przygotowywania danych testowych, narzędzi do projektowania testów i narzędzi do automatyzacji testów

TA-7.2.2 (K2) Kandydat potrafi wyjaśnić rolę Analityka Testów w automatyzacji testów opartej o słowa kluczowe

TA-7.2.3 (K2) Kandydat potrafi wyjaśnić kroki analizy problemu w przypadku niepowodzenia wykonywania testu automatycznego

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

7.1 Wprowadzenie

Narzędzia testowe mogą znacznie zwiększyć efektywność i dokładność testowania, jednak pod warunkiem, że będą to odpowiednie narzędzia zastosowane w odpowiedni sposób. Zarządzanie narzędziami testowymi jest jednym z aspektów dobrze zorganizowanego działu testów. Możliwości i zakres zastosowania narzędzi testowych bywają bardzo różne, a na tym rynku stale zachodzą zmiany. Narzędzia można zwykle pozyskać od producentów narzędzi komercyjnych lub z repozytoriów oprogramowania darmowego lub o ograniczonej licencji.

7.2 Narzędzia testowe i automatyzacja testów

Znaczna część prac wykonywanych przez Analityka Testów wymaga skutecznego wykorzystania narzędzi. Wiedza, jakie narzędzia zastosować i kiedy, może zwiększyć efektywność Analityka Testów i zapewnić lepsze pokrycie produktu testami w przewidzianym czasie.

7.2.1 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 (np. UML) lub z danych wprowadzanych przez Analityka Testów. Narzędzia do projektowania testów są często projektowane i tworzone pod kątem określonych formatów i produktów, na przykład konkretnych narzędzi do zarządzania wymaganiami.

Narzędzia do projektowania testów mogą dostarczać Analitykowi Testów informacji przydatnych do ustalenia typów testów, jakie są konieczne do osiągnięcia żądanego poziomu pokrycia produktu testami, zaufania do systemu lub łagodzenia ryzyka. Na przykład narzędzia drzew klasyfikacji generują (i wyświetlają) zestawy kombinacji, jakie zapewniają pełne pokrycie zgodnie z wybranym kryterium pokrycia. Na podstawie tych informacji Analityk Testów może następne ustalić, jakie przypadki testowe muszą zostać wykonane.

7.2.2 Narzędzia do przygotowywania danych testowych

Narzędzia do przygotowywania danych testowych przynoszą kilka korzyści. Niektóre takie narzędzia oferują możliwość przeanalizowania dokumentu takiego jak dokument wymagań czy nawet kod źródłowy w celu ustalenia danych wymaganych do osiągnięcia zadanego poziomu pokrycia produktu podczas testowania. Za pomocą innych narzędzi można pobrać zestaw danych z systemu produkcyjnego i „wyczyścić" go czy też „zanonimizować", tak aby usunąć z niego wszelkie dane osobowe, zachowując zarazem jego spójność wewnętrzną. 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. Jeszcze inne narzędzia do generowania danych testowych umożliwiają wygenerowanie danych testowych na podstawie zadanych 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.

7.2.3 Narzędzia do automatyzacji testów

Narzędzia do automatyzacji testów są używane na wszystkich poziomach testowania i zwykle umożliwiają Analitykom Testów wykonywanie testów i sprawdzanie ich rezultatów. Narzędzia do automatyzacji testów wykorzystuje się z reguły do 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).

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

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

7.2.3.1 Obszary zastosowania

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

Narzędzia do automatyzacji testów powszechnie wykorzystuje się na poziomie testów systemowych i integracyjnych. Niektóre narzędzia, zwłaszcza narzędzia do testowania interfejsów API, można stosować również na poziomie testów modułowych. Wykorzystanie narzędzi w testach tego rodzaju, do którego najlepiej się nadają, zwiększa zwrot z inwestycji.

7.2.3.2 Podstawowe informacje o narzędziach do automatyzacji testów

Działanie narzędzi do automatyzacji testów polega na wykonywaniu zestawów instrukcji zapisanych w języku programowania, często nazywanego językiem skryptowym. Instrukcje przekazywane do narzędzia są bardzo szczegółowe i uwzględniają definicje danych wejściowych, ich kolejność i konkretne wartości oraz odpowiednie oczekiwane dane wyjściowe. Taka szczegółowość skryptów powoduje, że są one wrażliwe na zmiany w przedmiocie testów, zwłaszcza gdy narzędzie wchodzi w interakcje z graficznym interfejsem użytkownika (GUI).

Większość narzędzi do automatyzacji testów zawiera komparator umożliwiający porównanie rzeczywistego rezultatu testu z zapisanym rezultatem oczekiwanym.

7.2.3.3 Implementacja automatyzacji testów

Trendem w automatyzacji testów (podobnie jak w programowaniu) jest odejście od szczegółowych instrukcji niskiego poziomu na rzecz języków wyższego poziomu i wykorzystanie bibliotek, makr i podprogramów. Techniki projektowania w oparciu o słowa kluczowe (słowa akcji) polegają na nagraniu szeregu instrukcji, a następnie wykorzystaniu tych spośród nich, które zawierają określone słowa kluczowe lub słowa akcji. Umożliwia to Analitykowi Testów pisanie przypadków testowych w języku naturalnym bez uciekania się do języka programowania i funkcji niskopoziomowych. Taka modułowa technika pisania testów ułatwia ich utrzymanie w przypadku zmian funkcjonalności i interfejsu przedmiotu testów. [Bath08] Więcej informacji o słowach kluczowych w skryptach automatycznych można znaleźć dalej w treści niniejszego dokumentu.

Przy tworzeniu słów kluczowych (słów akcji) można się posłużyć modelami. Na podstawie modeli procesów biznesowych, często zamieszczanych w dokumentacji wymagań, Analityk Testów może ustalić kluczowe procesy biznesowe, jakie muszą zostać przetestowane. Następnie można ustalić kroki tych procesów łącznie z punktami decyzji, jakie mogą zaistnieć w danym procesie. Punkty decyzji można przekształcić w słowa akcji, które narzędzie do automatyzacji testów następnie może pobrać z odpowiednich arkuszy. Modelowanie procesów biznesowych jest metodą dokumentowania procesów biznesowych i w jego toku można zidentyfikować kluczowe takie procesy i punkty decyzji. Modelowanie może być prowadzone ręcznie lub przy użyciu narzędzi, które działają w oparciu o dane wejściowe z informacjami o regułach biznesowych i opisami procesów.

7.2.3.4 Zwiększanie szans powodzenia automatyzacji

Przy wyborze testów, które mają zostać zautomatyzowane, należy ocenić każdy przypadek testowy lub zestaw testowy pod kątem sensu jego automatyzowania. Wiele projektów automatyzacji zakończonych

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

niepowodzeniem opiera się na automatyzowaniu istniejących manualnych przypadków testowych bez sprawdzenia, czy ich zautomatyzowanie rzeczywiście przyniesie jakieś korzyści. Dla danego zbioru przypadków testowych (zestawu testowego) optymalne może się okazać połączenie testów manualnych, częściowo zautomatyzowanych i w pełni automatycznych.

Podczas implementacji projektu automatyzacji testów należy uwzględnić następujące aspekty:

Potencjalne korzyści:

- czas wykonywania testów automatycznych łatwiej daje się przewidzieć,
- testy regresywne i sprawdzanie poprawek defektów za pomocą testów automatycznych jest szybsze i bardziej niezawodne w późnych fazach projektu,
- dzięki zastosowaniu narzędzi automatycznych może wzrosnąć status i wiedza techniczna testerów czy zespołu testowego,
- automatyzacja może być szczególnie przydatna w iteracyjnych i przyrostowych cyklach rozwoju oprogramowania dzięki zapewnieniu lepszych testów regresywnych każdej wersji czy iteracji,
- pokrycie niektórych typów testów jest możliwe tylko przy użyciu narzędzi do automatyzacji testów (np. walidacja dużych zbiorów danych),
- automatyzacja wykonywania testów może być bardziej ekonomiczną alternatywą dla testów manualnych w przypadku projektów wymagających wprowadzania, konwersji i porównywania dużych ilości danych ze względu na ich szybkie i spójne wprowadzanie i weryfikację.

Potencjalne ryzyko:

- niekompletne, nieefektywne lub niepoprawne testy manualne mogą zostać bezmyślnie zautomatyzowane,
- testalia mogą być trudne w pielęgnacji i wymagać licznych zmian wraz ze zmianami w testowanym oprogramowaniu,
- może zostać ograniczone bezpośrednie zaangażowanie testerów w wykonywanie testów, co poskutkuje mniejszą wykrywalnością defektów,
- zespół testowy może nie posiadać wystarczających umiejętności, aby efektywnie korzystać z narzędzi do automatyzacji,
- nieistotne testy, które nie przyczyniają się do zwiększenia ogólnego pokrycia, mogą zostać zautomatyzowane tylko dlatego, że istnieją i są stabilne,
- testy mogą stać się bezproduktywne wraz ze stabilizacją oprogramowania (paradoks pestycydów).

Bezpośrednie zautomatyzowanie manualnych przypadków testowych nie zawsze jest dobrym rozwiązaniem podczas wdrożenia narzędzia do automatyzacji testów; często warto przedefiniować te przypadki testowe w celu lepszego wykorzystania możliwości oferowanych przez automatyzację. Obejmuje to formatowanie przypadków testowych, rozważenie wzorców do ponownego wykorzystania, rozbudowanie danych wejściowych poprzez zastosowanie zmiennych zamiast zapisanych sztywno wartości i pełne wykorzystanie możliwości płynących z zastosowania narzędzia do testów. Narzędzia do automatyzacji testów mogą zwykle wykonać wiele testów jeden po drugim, grupować testy, powtarzać je i zmieniać kolejność ich wykonywania, jednocześnie udostępniając funkcje analizy i raportowania.

W przypadku wielu narzędzi automatyzacji testów do tworzenia skutecznych i efektywnych testów (skryptów) i zestawów testowych konieczna jest umiejętność programowania. Aktualizowanie dużych zestawów testów automatycznych i zarządzanie nimi może być bardzo trudne, jeżeli nie są one dobrze zaprojektowane. Pełne wykorzystanie możliwości płynących z zastosowania narzędzi ułatwia odpowiednie przeszkolenie w zakresie korzystania z narzędzi testowych, programowania i technik projektowania.

Podczas planowania testów należy przewidzieć czas na regularne wykonywanie automatycznych przypadków testowych manualnie, aby zachować wiedzę o zawartości testu i weryfikować jego prawidłowe działanie oraz poprawność danych wejściowych i zapewniane przez nie pokrycie.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

7.2.3.5 Automatyzacja oparta o słowa kluczowe

Słów kluczowych (czasem nazywanych słowami akcji) używa się z reguły, choć nie tylko, jako reprezentacji interakcji biznesowych z systemem na wysokim poziomie (np. „wycofaj zamówienie"). Każde słowo kluczowe reprezentuje zwykle szereg szczegółowych interakcji między pewnym aktorem a testowanym systemem. Przypadki testowe specyfikuje się za pomocą sekwencji słów kluczowych (wraz z odpowiednimi danymi testowymi). [Buwalda01]

Podczas automatyzowania testu słowo kluczowe implementuje się w postaci jednego lub kilku skryptów testowych. Narzędzia odczytują przypadki testowe zapisane jako sekwencje słów kluczowych, które stanowią wywołania odpowiednich skryptów testowych z implementacją odpowiednich funkcjonalności. Skrypty są zbudowane modułowo, aby łatwo je było odwzorowywać na konkretne słowa kluczowe. Do zaimplementowania takich modułowych skryptów konieczna jest umiejętność programowania.

Podstawowe korzyści płynące z automatyzacji testów w oparciu o słowa kluczowe 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 posiadająca głównie wiedzę dziedzinową może wykorzystać automatycznie wykonywane przypadki testowe (po tym, jak słowa kluczowe zostały zaimplementowane w postaci skryptów) bez konieczności rozumienia kodu automatyzacji,
- przypadki testowe zapisane za pomocą słów kluczowych są łatwiejsze w pielęgnacji, gdyż konieczność ich modyfikowania w przypadku zmian w testowanym oprogramowaniu jest mniej prawdopodobna,
- specyfikacja przypadków testowych jest niezależna od ich implementacji; słowa kluczowe można implementować za pomocą różnych narzędzi i języków skryptowych.

Tworzenie skryptów automatyzacji (faktycznego kodu automatyzacji) wykorzystujących informacje o słowach kluczowych / słowach akcji jest zwykle zadaniem programistów lub Technicznych Analityków Testów, podczas gdy Analitycy Testów z reguły tworzą i pielęgnują dane o słowach kluczowych / słowach akcji. Automatyzacja oparta o słowa kluczowe odbywa się z reguły w fazie testów systemowych, ale prace nad kodem mogą się rozpocząć już w fazie integracji. W środowisku iteracyjnym prace nad automatyzacją testów są procesem ciągłym.

Po utworzeniu wejściowych słów kluczowych i danych Analityk Testów przejmuje zwykle odpowiedzialność za wykonanie przypadków testowych opartych na słowach kluczowych i za analizę wszelkich napotkanych awarii. W przypadku wykrycia anomalii Analityk Testów musi zbadać przyczynę awarii, aby stwierdzić, czy problemem są słowa kluczowe, dane wejściowe, skrypt automatyzacji czy wreszcie 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 wygenerowane niepoprawne dane), a ujawnił się dopiero później w toku przetwarzania. Jeżeli Analityk Testów nie jest w stanie ustalić przyczyny awarii, powinien przekazać zgromadzone podczas analizy problemu informacje Technicznemu Analitykowi Testów lub programiście do dalszego zbadania.

7.2.3.6 Przyczyny niepowodzenia projektów automatyzacji

Projekty automatyzacji wykonywania testów często nie osiągają zamierzonych celów. Te niepowodzenia mogą wynikać z niewystarczającej elastyczności w posługiwaniu się narzędziem testowym, niewystarczających umiejętnościach programistycznych zespołu testowego lub nierealistycznych wyobrażeń co do problemów, jakie może rozwiązać automatyzacja testów. Należy zauważyć, że wszelkie próby automatyzacji wykonywania testów wymagają, jak każdy inny projekt programistyczny, odpowiedniego zarządzania, nakładu pracy, odpowiednich umiejętności i wystarczającej uwagi. Należy przeznaczyć odpowiednią ilość czasu na zbudowanie architektury umożliwiającej długofalowy rozwój, zgodności z przyjętymi praktykami projektowania, na zarządzanie konfiguracją i działania zgodne z przyjętymi praktykami tworzenia kodu. Skrypty testów automatycznych muszą zostać przetestowane,

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

ponieważ prawdopodobnie zawierają defekty. Może być konieczne ich dostrojenie w celu uzyskania większej wydajności. Należy wziąć pod uwagę użyteczność narzędzia zarówno dla programisty, jak i dla osób, które będą korzystać z tego narzędzia do wykonywania skryptów. Może być konieczne zaprojektowanie takiego interfejsu między narzędziem a testerem, który udostępni testerowi przypadki testowe w logicznie zorganizowanej formie, ale zarazem zapewni ich dostępność w postaci wymaganej dla narzędzia.

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

8. Dokumenty pomocnicze

8.1 Normy

- [ISO25000] ISO/IEC 25000:2005, Software Engineering — Software Product Quality Requirements and Evaluation (SQuaRE)
  rozdziały 1 i 4
- [ISO9126] ISO/IEC 9126-1:2001, Software Engineering — Software Product Quality,
  rozdziały 1 i 4
- [RTCA DO-178B/ED-12B]: Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12B.1992.
  rozdział 1

8.2 Dokumenty ISTQB

- [ISTQB_AL_OVIEW] Wprowadzenie do ISTQB poziomu zaawansowanego, wersja 1.0
- [ISTQB_ALTM_SYL] ISTQB sylabus poziomu zaawansowanego — Kierownik Testów, wersja 1.0
- [ISTQB_ALTTA_SYL] ISTQB sylabus poziomu zaawansowanego — Techniczny Analityk Testów, wersja 1.0
- [ISTQB_FL_SYL] ISTQB sylabus poziomu podstawowego, wersja 2011
- [ISTQB_GLOSSARY] Standardowy słownik terminów używanych w testowaniu oprogramowania, wersja 2.2, 2012

8.3 Książki

[Bath08]: Graham Bath, Judy McKay, „The Software Test Engineer's Handbook", Rocky Nook, 2008, 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"*, John Wiley and Sons, 2007, ISBN 978-0-470-12790-2

[Buwalda01]: Hans Buwalda, „Integrated Test Design and Automation", Addison-Wesley Longman, 2001, ISBN 0-201-73725-6

[Cohn04]: Mike Cohn, „User Stories Applied: For Agile Software Development", Addison-Wesley Professional, 2004, ISBN 0-321-20568-5

[Copeland03]: Lee Copeland, „A Practitioner's Guide to Software Test Design", Artech House, 2003, ISBN 1-58053-791-X

[Craig02]: Rick David Craig, Stefan P. Jaskiel, „*Systematic Software Testing"*, Artech House, 2002, ISBN 1-580-53508-9

[Gerrard02]: Paul Gerrard, Neil Thompson, „Risk-based e-business Testing", Artech House, 2002, ISBN 1-580-53314-0

[Gilb93]: Tom Gilb, Dorothy Graham, „Software Inspection", Addison-Wesley, 1993, ISBN 0-201-63181-4

[Graham07]: Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black „*Foundations of Software Testing",* Thomson Learning, 2007, ISBN 978-1-84480-355-2

[Grochmann94]: M. Grochmann (1994), Test case design using Classification Trees, in: conference proceedings STAR 1994

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

[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

[Myers79]: Glenford J. Myers, „The Art of Software Testing”, John Wiley & Sons, 1979, ISBN 0-471-46912-2

[Splaine01]: Steven Splaine, Stefan P. Jaskiel, „The Web-Testing Handbook”, STQE Publishing, 2001, ISBN 0-970-43630-0

[vanVeenendaal12]: Erik van Veenendaal, „Practical risk-based testing – The PRISMA approach”, UTN Publishers, The Netherlands, ISBN 9789490986070

[Wiegers03]: Karl Wiegers, „Software Requirements 2”, 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”, Addison-Wesley, 2009, ISBN 0-321-63641-4

8.4 Inne dokumenty pomocnicze

Następujące odwołania wskazują informacje dostępne w Internecie i w innych źródłach. Te 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
  - taksonomia defektów: www.testingeducation.org/a/bsct2.pdf
  - przykładowa taksonomia defektów oparta na pracach Borisa Beizera: inet.uni2.dk/~vinter/bugtaxst.doc
  - dobry przegląd różnych taksonomii: testingeducation.org/a/bugtax.pdfhttp://testingeducation.org/a/bugtax.pdf
  - Bach, James: Heuristic Risk-Based Testing
  - z „Exploratory & Risk-Based Testing (2004) www.testingeducation.org”
  - Exploring Exploratory Testing, Cem Kaner i Andy Tikam, 2003
  - Pettichord, Bret, „An Exploratory Testing Workshop Report”, www.testingcraft.com/exploratorypettichordhttp://www.testingcraft.com/exploratorypettichord
- rozdział 4
  - www.testingstandards.co.uk

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

9. Indeks

0-przełączenie, 32
analiza dziedzinowa, 27, 36
analiza przyczyny podstawowej, 55, 58
analiza ryzyka, 21
analiza testów, 12
analiza wartości brzegowych, 27, 29
ankiety, 49
anonimizować, 61
atrakcyjność, 43
automatyzacja
  korzyści z, 63
  ryzyko, 63
automatyzacja oparta na słowach kluczowych, 64
charakterystyka podrzędna jakości, 44
charakterystyki jakości, 44
cykl życia oprogramowania, 9
cykl życia wytwarzania oprogramowania
  iteracyjny, 10
  zwinny, 10
czynności, 10
czynności związane z zakończeniem testów, 20
defekt
  klasyfikacja, 57
  pola, 56
  taksonomia, 55
  wykrywanie, 56
dokładność, 43
dopasowanie, 43
dostępność, 43
drzewa klasyfikacji, 27, 33
funkcjonalne charakterystyki jakości, 45
heurystyka, 43, 48
historyjki użytkownika, 15, 31, 35, 52, 53
identyfikacja czynników ryzyka, 21, 24
implementacja testów, 8, 16
incydent, 18
inspekcja, 48
ISO 25000, 15, 44
ISO 9126, 15, 44
karta opisu testu, 27
klasy równoważności, 27, 28
konkretne przypadki testowe, 13, 14
konkretny przypadek testowy, 8
kryteria wyjścia, 8
kwestionariusze, 49
łączenie technik, 37
łagodzenie ryzyka, 21, 22, 25
łatwość nauki, 43
łatwość obsługi, 43
listy kontrolne do wymagań, 52
listy kontrolne w przeglądach, 51
logiczne przypadki testowe, 14
logiczny przypadek testowy, 8
logowanie testów, 18
metryki, 12
modelowanie procesów biznesowych, 62
monitorowanie i nadzór, 12
monitorowanie i nadzorowanie postępu testów, 22
monitorowanie testów, 21
nadzór nad testami, 8
narzędzia, 61
  do automatyzacji testów, 60, 61
  do projektowania testów, 60, 61
  do przygotowywania danych testowych, 60, 61
narzędzie projektowania testów, 31
niefunkcjonalne charakterystyki jakości, 46
nietestowalne, 52
ocena, 48
ocena kryteriów wyjścia i raportowanie, 19
ocena ryzyka, 24
oparte na słowach akcji, 62
oparte na słowach kluczowych, 60, 62
oszacowania testów, 11
paradoks pestycydów, 17
plan testów, 11
planowanie testów, 8, 11
podejście głębokie, 26
podejście szerokie, 26
podstawa testów, 14, 15
pokrycie N-przełączeń, 33
powstrzymanie fazowe, 55, 56
poziom ryzyka, 21
projekt testów, 8
projektowanie testów, 13
prototyp, 49
przegląd, 48, 51
przełączenie N-1, 32
przyczyna podstawowa, 12, 57, 58
przypadek testowy, 14
przypadek testowy niskiego poziomu, 8
przypadek testowy wysokiego poziomu, 8
rezultat fałszywie negatywny, 18
rezultat fałszywie pozytywny, 18
ryzyko produktowe, 13, 21
śledzenie, 12
śledzenie defektów, 22
słowa akcji, 64
specyfikacja testów użyteczności, 48
spotkania retrospektywne, 20
środowisko testowe, 17
standardy

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych

DO-178B, 16
ED-12B, 16
UML, 49
strategia testów, 12, 14, 21
strategia testów oparta na ryzyku, 16
SUMI, 43, 49
tablica ortogonalna, 27, 33
tablice decyzyjne, 30
taksonomia defektów, 27, 38, 55
technika oparta na defektach, 27
technika oparta na doświadczeniu, 27
technika oparta na specyfikacji, 27
techniki oparte na defektach, 37
techniki oparte na doświadczeniu, 17, 27, 39, 42
techniki oparte na specyfikacji, 28
techniki testowania, 27
techniki testowania kombinatoryjnego, 33
testowanie charakterystyk jakości oprogramowania, 43
testowanie dokładności, 46
testowanie dopasowania, 46
testowanie dostępności, 49
testowanie eksploracyjne, 27, 41
testowanie funkcjonalne, 45
testowanie kombinatoryjne, 27, 33, 47
testowanie sposobem par, 27
testowanie oparte na ryzyku, 21, 24
testowanie oparte na wymaganiach, 27
testowanie przejść pomiędzy stanami, 27, 32
testowanie rozproszone, 23
testowanie rozproszone, zlecone na zewnątrz, zlecone wewnątrz, 23
testowanie scentralizowane, 23
testowanie użyteczności, 47
testowanie w oparciu o historyjki użytkownika, 27, 35
testowanie w oparciu o listę kontrolną, 27, 40
testowanie w oparciu o przypadki użycia, 27, 34
testowanie w oparciu o tablicę decyzyjną, 27
testowanie w oparciu o tablice ortogonalne, 33
testowanie współdziałania, 46
testowanie zlecone na zewnątrz, 23
testowanie zlecone wewnątrz, 23
tworzenie grafów przyczynowo-skutkowych, 27, 31
użyteczność, 43
walidacja, 49
WAMMI, 43, 49
warunki testowe, 12
wbudowany iteracyjny, 10
współdziałanie, 43
wybór najlepszej techniki, 42
wykonanie testów, 8, 17
wyrocznia testowa, 14
zarządzanie ryzykiem, 21
zestaw testów regresywnych, 20
zestawy testowe, 16
zgadywanie błędów, 27, 39
zgodność, 44
zrozumiałość, 43
zwinne, 10, 15, 23, 35, 45, 53, 66

International Software Testing Qualifications Board® © Stowarzyszenie Jakości Systemów Informatycznych