Certyfikowany tester ISTQB®

Sylabus poziomu zaawansowanego Techniczny Analityk Testów (TTA)

wersja 4.0

International Software Testing Qualifications Board®

Prawa autorskie

Tłumaczenie z języka angielskiego wersji beta – BTInfo Biuro Tłumaczeń Informatycznych Przyłuccy sp. j.
Przeglądy i uaktualnienie do wersji 4.0 przygotował zespół SJSI w składzie: Adam Roman (kierownik zespołu), Lucjan Stapp, Arnika Hryszko.
Redakcja tekstu i korekta tłumaczenia: Monika Petri-Starego.

Historia zmian

Wersja	Data	Uwagi
Wersja 4.0	30 czerwca 2021 r.	Wersja 4.0 zatwierdzona do publikacji przez Zgromadzenie Ogólne ISTQB®
Wersja 4.0	28 kwietnia 2021 r.	Aktualizacja dokumentu na podstawie uwag z przeglądu beta
Wersja 4.0 beta	1 marca 2021 r.	Aktualizacja dokumentu na podstawie uwag z przeglądu alfa
Wersja 4.0 alfa	7 grudnia 2020 r.	Szkic wersji alfa z następującymi zmianami: • Poprawki w tekście • Usunięcie sekcji związanej z K3 TTA-2.6.1. (testowanie ścieżek) oraz usunięcie celu nauczania • Usunięcie sekcji związanej z K2 TTA-3.2.4 (grafy wywołań) oraz usunięcie celu nauczania • Zmiany w sekcji 3.2.2. (TTA-3.2.2., analiza przepływu danych) i podniesienie z K2 na K3 • Przepisanie podrozdziału 4.4. związanej z TTA-4.4.1. oraz TTA-4.4.2. (testowanie niezawodności) • Przepisanie podrozdziału 4.5. związanego z TTA-4.5.1. oraz TTA-4.5.2. (testowanie wydajnościowe) • Dodanie podrozdziału 4.9. dotyczącego profili operacyjnych • Przepisanie podrozdziału 2.7. związanego z TTA-2.8.1. (wybór technik białoskrzynkowych) • Modyfikacja TTA-3.2.1. – dodanie informacji o złożoności cyklomatycznej (brak wpływu na pytania egzaminacyjne) • Przepisanie sekcji związanej z TTA-2.4.1. (MC/DC), aby uspójnić ją z innymi technikami białoskrzynkowymi (brak wpływu na pytania egzaminacyjne)
Wersja 2019 1.0	18 października 2019 r.	Wersja 2019 zatwierdzona do publikacji przez Zgromadzenie Ogólne ISTQB®
Wersja 2012	19 października 2012 r.	Wersja 2012 zatwierdzona do publikacji przez Zgromadzenie Ogólne ISTQB®

Historia zmian wersji polskiej

Wersja	Data	Uwagi
Wersja 4.0.	15.07.2021 r. 15.09.2021 r.	Przegląd/aktualizacja na podstawie tłumaczenia wersji 2019 oraz zmian wprowadzonych w wersji 4.0.
Wersja 1.0.	01.04.2020 r. – 30.05.2020 r.	Przegląd tłumaczenia – Zespół SJSI
Wersja 0.1.	01.03.2020 r. –31.03.2020 r.	Tłumaczenie wersji beta BTInfo Biuro Tłumaczeń Informatycznych Przyłuccy sp. j.

Spis treści

Podziękowania

Wersja 2019 niniejszego dokumentu została opracowana przez podstawowy zespół Grupy Roboczej ds. Poziomu Zaawansowanego (Advanced Level Working Group) działający w ramach ISTQB® w następującym składzie: Graham Bath (wiceprzewodniczący), Rex Black, Judy McKay, Kenji Onoshi, Mike Smith (przewodniczący), Erik van Veenendaal.

W procesie weryfikacji, zgłaszania uwag i głosowania nad niniejszym sylabusem uczestniczyli:

Dani Almog	Andrew Archer	Rex Black
Armin Born	Sudeep Chatterjee	Tibor Csöndes
Wim Decoutere	Klaudia Dusser-Zieger	Melinda Eckrich-Brájer
Peter Foldhazi	David Frei	Karol Frühauf
Jan Giesen	Attila Gyuri	Matthias Hamburg
Tamás Horváth	N. Khimanand	Jan te Kock
Attila Kovács	Claire Lohr	Rik Marselis
Marton Matyas	Judy McKay	Dénes Medzihradszky
Petr Neugebauer	Ingvar Nordström	Pálma Polyák
Meile Posthuma	Stuart Reid	Lloyd Roden
Adam Roman	Jan Sabak	Péter Sótér
Benjamin Timmermans	Stephanie van Dijck	Paul Weymouth

Niniejszy dokument został opracowany przez Grupę Roboczą ds. Poziomu Zaawansowanego (Advanced Level Working Group) działającą w ramach ISTQB® w składzie: Armin Born, Adam Roman, Stuart Reid.

Zaktualizowana wersja niniejszego dokumentu została opracowana przez Grupę Roboczą ds. Poziomu Zaawansowanego (Advanced Level Working Group) działającą w ramach ISTQB® w składzie: Armin Born, Adam Roman, Christian Graf, Stuart Reid.

W przeglądzie, komentowaniu i głosowaniu nad zaktualizowaną wersją 4.0. niniejszego sylabusa uczestniczyli:

Ágota Horváth	Lloyd Roden	Paul Weymouth
Benjamin Timmermans	Matthias Hamburg	Péter Földházi Jr.
Erwin Engelsma	Meile Posthuma	Rik Marselis
Gary Mogyorodi	Nishan Portoyan	Sebastian Małyska
Geng Chen	Joan Killeen	Tal Pe'er
Gergely Ágnecz	Ole Chr. Hansen	Vécsey-Juhász Adél
Jane Nash	Pálma Polyák	Wang Lijuan
Zuo Zhenlei.		

Grupa Robocza ds. Poziomu Zaawansowanego składa podziękowania zespołowi weryfikatorów oraz Radom Krajowym za sugestie i wskazówki.

Niniejszy dokument został formalnie wydany przez Zgromadzenie Ogólne ISTQB® 30 czerwca 2021 r.

0. Wprowadzenie

0.1 Cel sylabusa

Niniejszy sylabus stanowi podstawę egzaminu International Software Testing Qualifications Board dla Technicznego Analityka Testów na poziomie zaawansowanym. ISTQB® udostępnia sylabus:

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 przygotowania pytań egzaminacyjnych w języku lokalnym, odpowiadających celom nauczania sylabusa.
3. Dostawcom szkoleń w celu opracowania materiałów dydaktycznych i określenia odpowiednich metod nauczania.
4. Kandydatom ubiegającym się o certyfikat w celu przygotowania do egzaminu (w ramach szkoleń zorganizowanych lub samodzielnie).
5. Międzynarodowej społeczności specjalistów w dziedzinie inżynierii oprogramowania i systemów w celu rozwijania zawodu testera oprogramowania i systemów oraz jako podstawę do opracowywania książek i artykułów.

ISTQB® może zezwolić innym podmiotom na korzystanie z niniejszego sylabusa do innych celów, o ile te podmioty uprzednio złożą wniosek o pisemne zezwolenie na takie korzystanie i otrzymają je.

0.2 Certyfikowany tester — poziom zaawansowany w testowaniu oprogramowania

Kwalifikacja na poziomie zaawansowanym obejmuje trzy odrębne sylabusy związane z następującymi rolami:

- Kierownik Testów,
- Analityk Testów,
- Techniczny Analityk Testów.

„Certyfikowany tester ISTQB®, Sylabus poziomu zaawansowanego Techniczny Analityk Testów (TTA) Omówienie sylabusa wersja 4.0” to oddzielny dokument [ISTQB_AL_TTA_OVERIEW], w którym zawarto następujące informacje:

- rezultaty biznesowe,
- macierz powiązań między rezultatami biznesowymi a celami nauczania,
- podsumowanie.

0.3 Cele nauczania objęte egzaminem i poziomy poznawcze

Cele nauczania wspierają osiąganie rezultatów biznesowych i służą do przygotowania egzaminów do uzyskania certyfikatu „Techniczny Analityk Testów — poziom zaawansowany”. Poziomy wiedzy związane z poszczególnymi celami nauczania przedstawiono na początku każdego rozdziału. Poziomy te sklasyfikowano następująco:

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

0.4 Oczekiwane doświadczenie

Niektóre z celów nauczania określonych dla Technicznego Analityka Testów zakładają posiadanie podstawowej wiedzy w następujących obszarach:

- ogólne koncepcje dotyczące programowania,
- ogólne koncepcje dotyczące architektury systemów.

0.5 Egzamin na poziomie zaawansowanym dla Technicznego Analityka Testów

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

Egzamin ma formę testu wielokrotnego wyboru i składa się z 45 pytań. Do zdania egzaminu niezbędne jest uzyskanie co najmniej 65% punktów.

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

0.6 Wymagania stawiane kandydatom przystępującym do egzaminu

Przed przystąpieniem do egzaminu certyfikacyjnego dla Technicznego Analityka Testów na poziomie zaawansowanym należy zdać egzamin certyfikacyjny związany z sylabusem Certyfikowany Tester - Poziom Podstawowy.

0.7 Akredytacja szkoleń

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

0.8 Poziom szczegółowości informacji

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

- ogólne cele dydaktyczne opisujące założenia poziomu zaawansowanego w odniesieniu do Technicznego Analityka Testów,
- wykaz terminów, które muszą zapamiętać uczestnicy szkolenia,
- cele nauczania w poszczególnych obszarach wiedzy, opisujące rezultaty kształcenia o charakterze poznawczym,
- opis najważniejszych pojęć, w tym odsyłacze do źródeł (takich jak uznane publikacje oraz normy lub standardy).

Treść sylabusa nie stanowi opisu całego obszaru wiedzy związanego z testowaniem oprogramowania. Odzwierciedla ona jedynie poziom szczegółowości, jaki należy uwzględnić w akredytowanych szkoleniach na poziomie zaawansowanym. Sylabus koncentruje się na zagadnieniach, które mogą dotyczyć dowolnych projektów wytwarzania oprogramowania i dowolnego cyklu życia oprogramowania. Sylabus nie zawiera żadnych konkretnych celów nauczania związanych z określonym modelem wytwarzania oprogramowania, natomiast omówiono w nim, w jaki sposób wprowadzone pojęcia można zastosować do modelu zwinnego wytwarzania oprogramowania, do innych modeli iteracyjnych i przyrostowych oraz do modeli sekwencyjnych.

0.9 Struktura sylabusa

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

- Rozdział 1.: Zadania Technicznego Analityka Testów w testowaniu opartym na ryzyku (30 minut),
- Rozdział 2.: Białoskrzynkowe techniki testowania (300 minut),
- Rozdział 3.: Analiza statyczna i dynamiczna (180 minut),
- Rozdział 4.: Charakterystyki jakościowe w testach technicznych (345 minut),
- Rozdział 5.: Przeglądy (165 minut),
- Rozdział 6.: Narzędzia testowe i automatyzacja testów (180 minut).

1. Zadania Technicznego Analityka Testów w testowaniu opartym na ryzyku — 30 minut

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

Cele nauczania związane z zadaniami Technicznego Analityka Testów w testowaniu opartym na ryzyku

1.2. Zadania związane z testowaniem opartym na ryzyku

TTA-1.2.1. (K2) Kandydat potrafi omówić ogólne czynniki ryzyka, które zwykle musi wziąć pod uwagę Techniczny Analityk Testów

TTA-1.2.2. (K2) Kandydat potrafi omówić czynności wykonywane przez Technicznego Analityka Testów w ramach podejścia do testowania opartego na ryzyku, związane z czynnościami testowymi

1.1 Wstęp

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

Techniczny Analityk Testów działa w ramach struktury testowania (ang. *testing framework*) opartego na ryzyku zbudowanej przez Kierownika Testów dla potrzeb danego projektu. Wnoszą oni swoją wiedzę na temat technicznych ryzyk produktowych, które są nieodłącznym elementem projektu, takich jak ryzyka dotyczące zabezpieczeń, niezawodności systemu i wydajności. Powinni również wnosić wkład w identyfikację i sposób traktowania ryzyk projektowych związanych ze środowiskami testowymi, dotyczącymi np. pozyskania i konfigurowania środowisk testowych dla testów wydajnościowych, niezawodności czy zabezpieczeń.

1.2 Zadania związane z testowaniem opartym na ryzyku

Techniczni analitycy testów aktywnie uczestniczą w następujących zadaniach związanych z testowaniem opartym na ryzyku:

- identyfikacja ryzyka,
- ocena ryzyka,
- łagodzenie ryzyka.

Zadania te są wykonywane iteracyjnie w trakcie całego projektu, tak, aby zespół projektowy mógł radzić sobie z pojawiającymi się zagrożeniami i zmieniającymi się priorytetami oraz regularnie oceniać status ryzyka i informować o nim interesariuszy.

1.2.1 Identyfikacja ryzyka

W procesie identyfikacji ryzyka szanse na wykrycie jak największej liczby potencjalnych istotnych czynników ryzyka są tym większe, im większa grupa interesariuszy weźmie w nim udział. Techniczni analitycy testów dysponują unikalnymi kompetencjami technicznymi, więc są szczególnie predysponowani do przeprowadzania wywiadów z ekspertami, prowadzenia sesji „burz mózgów” ze współpracownikami oraz analizowania ich doświadczeń w celu określenia prawdopodobnych obszarów ryzyka produktowego. W szczególności techniczni analitycy testów ściśle współpracują z innymi interesariuszami, np. programistami, architektami, specjalistami ds. eksploatacji, właścicielami produktów, lokalnymi jednostkami wsparcia, ekspertami technicznymi i technikami serwisowymi, aby określić obszary ryzyka technicznego mające wpływ na dany produkt i projekt. Dzięki zaangażowaniu innych grup interesariuszy można uwzględnić wszystkie punkty widzenia. Koordynacją takiej współpracy zajmuje się zwykle Kierownik Testów.

Czynniki ryzyka, które może identyfikować Techniczny Analityk Testów, na ogół oparte są na charakterystykach jakościowych produktu określonych w standardzie [ISO 25010] i wyszczególnionych w Rozdziale 4. tego sylabusa.

1.2.2 Ocena ryzyka

Celem identyfikacji ryzyka jest znalezienie jak największej liczby istotnych czynników ryzyka; przedmiotem oceny ryzyka jest natomiast analiza tych czynników zmierzająca do sklasyfikowania każdego ryzyka oraz ustalenia prawdopodobieństwa wystąpienia i wpływu każdego z nich.

Przez prawdopodobieństwo wystąpienia ryzyka produktowego rozumie się zwykle prawdopodobieństwo wystąpienia awarii w testowanym systemie. Techniczny Analityk Testów wnosi wkład w zrozumienie prawdopodobieństwa wystąpienia każdego ryzyka produktowego, podczas gdy Analityk Testów wnosi wkład w zrozumienie potencjalnego wpływu biznesowego wystąpienia danego problemu.

Wystąpienie ryzyk projektowych może negatywnie wpłynąć na powodzenie całego projektu. W typowych sytuacjach należy rozważyć następujące ogólne czynniki ryzyka projektowego:

- konflikt między interesariuszami dotyczący wymagań technicznych,
- problemy w komunikacji wynikające z rozproszenia geograficznego jednostek organizacyjnych odpowiedzialnych za tworzenie oprogramowania,
- narzędzia i technologie (w tym istotne umiejętności),
- presję czasu, ograniczone zasoby i naciski ze strony kierownictwa,
- brak wcześniejszego zapewnienia jakości,
- dużą szybkość zmian wymagań technicznych.

Wystąpienie ryzyk produktowych może prowadzić do zwiększenia liczby defektów. W typowych sytuacjach należy rozważyć następujące ogólne czynniki ryzyka produktowego:

- złożoność technologii,
- złożoność kodu,
- ilość zmian w kodzie (dodanie, usunięcie, modyfikacja),
- duża liczba znalezionych defektów związanych z technicznymi charakterystykami jakościowymi (historia defektów),
- zagadnienia techniczne związane z interfejsami i integracją.

Na podstawie dostępnych informacji o ryzyku Techniczny Analityk Testów przedstawia propozycję początkowego prawdopodobieństwa ryzyka zgodnie z wytycznymi otrzymanymi od Kierownika Testów. Wartość początkowa może zostać zmodyfikowana przez Kierownika Testów po uwzględnieniu opinii wszystkich interesariuszy. Wpływ ryzyka jest zazwyczaj określany przez Analityka Testów.

1.2.3 Łagodzenie ryzyka

W czasie trwania projektu Techniczny Analityk Testów wpływa na to, jak w procesie testowania uwzględniane są zidentyfikowane czynniki ryzyka. Obejmuje to zwykle następujące elementy:

- Projektowanie przypadków testowych związanych z obszarami wysokiego ryzyka i pomoc w ocenie ryzyka rezydualnego.
- Redukcję ryzyka poprzez wykonanie zaprojektowanych przypadków testowych i wprowadzenie odpowiednich środków łagodzących i awaryjnych zgodnie z planem testów.
- Ocenę czynników ryzyka w oparciu o dodatkowe informacje zgromadzone w toku projektu oraz wykorzystanie tych informacji do wdrożenia środków związanych z łagodzeniem ryzyka, zmierzających do zmniejszenia prawdopodobieństwa wystąpienia poszczególnych czynników.

Techniczny Analityk Testów często współpracuje ze specjalistami, np. w dziedzinie zabezpieczeń i wydajności, aby zdefiniować środki łagodzenia ryzyka i elementy strategii testów. Dodatkowe informacje na ten temat można znaleźć w specjalistycznych sylabusach ISTQB®, np. w sylabusie Certyfikowany Tester – Tester Zabezpieczeń [CT_SEC_SYL] oraz sylabusie Certyfikowany Tester – Tester Wydajności [CT_PT_SYL].

2. Białoskrzynkowe techniki testowania — 300 minut

Słowa kluczowe
białoskrzynkowa technika testowania, poziom nienaruszalności bezpieczeństwa, testowanie decyzji, testowanie instrukcji, testowanie API, przepływ sterowania, testowanie warunków wielokrotnych, warunek atomowy, zmodyfikowane testowanie warunkowo-decyzyjne MC/DC[1]

Cele nauczania związane z białoskrzynkowymi technikami testowania

2.2. Testowanie instrukcji

TTA-2.2.1. (K3) Kandydat potrafi zaprojektować przypadki testowe dla podanego przedmiotu testów, korzystając z techniki testowania instrukcji, aby osiągnąć zdefiniowany poziom pokrycia

2.3. Testowanie decyzji

TTA-2.3.1. (K3) Kandydat potrafi zaprojektować przypadki testowe dla podanego przedmiotu testów korzystając z techniki testowania decyzji, aby osiągnąć zdefiniowany poziom pokrycia

2.4. Testowanie MC/DC

TTA-2.4.1. (K3) Kandydat potrafi zaprojektować przypadki testowe dla podanego przedmiotu testów korzystając z techniki testowania MC/DC, aby osiągnąć pełne pokrycie MC/DC

2.5. Testowanie warunków wielokrotnych

TTA-2.5.1. (K3) Kandydat potrafi zaprojektować przypadki testowe dla podanego przedmiotu testów korzystając z techniki testowania warunków wielokrotnych, aby osiągnąć zdefiniowany poziom pokrycia

2.6. Testowanie ścieżek podstawowych
(*podrozdział usunięty począwszy od wersji 4.0 niniejszego sylabusa*)
Cel nauczania TTA-2.6.1. został usunięty z niniejszej wersji sylabusa

2.7. Testowanie API

TTA-2.7.1. (K2) Kandydat rozumie obszary zastosowania testów API i rodzaje defektów wykrywanych w takich testach

2.8. Wybór białoskrzynkowej techniki testowania

TTA-2.8.1. (K4) Kandydat potrafi wybrać odpowiednią białoskrzynkową technikę testowania zgodnie z daną sytuacją projektową

[1] w dalszej części sylabusa będą używane terminy: "testowanie MC/DC" oraz "pokrycie MC/DC" jako terminy powszechnie używane

2.1 Wstęp

Niniejszy rozdział jest poświęcony białoskrzynkowym technikom testowania. Techniki te mają zastosowanie do kodu i innych struktur z przepływem sterowania, np. diagramów przepływu procesów biznesowych.

Poszczególne techniki umożliwiają systematyczne wyprowadzanie przypadków testowych i koncentrują się na konkretnych aspektach struktury. Przypadki testowe tworzone przy pomocy tych technik spełniają kryteria pokrycia ustanowione jako cel testowania. Pokrycie jest mierzone i odnoszone do tych celów. Uzyskanie pełnego (tzn. 100%) pokrycia nie oznacza, że zbiór testów jest kompletny, ale raczej to, że w ramach używanej techniki nie można już opracować dalszych przydatnych testów badanej struktury.

Dane wejściowe do testów są generowane tak, aby wykonanie testu zapewniło sprawdzenie określonej części kodu (np. instrukcji, wyników decyzji). Określenie danych wejściowych testu, które spowodują wykonanie określonej części kodu, może być trudne, zwłaszcza wtedy, gdy część kodu, który ma być wykonany, znajduje się na końcu długiej ścieżki przepływu sterowania zawierającej po drodze kilka punktów decyzyjnych. Oczekiwane wyniki testów są identyfikowane na podstawie źródła zewnętrznego wobec tej struktury, np. na podstawie wymagań, specyfikacji projektowej lub innej podstawy testów.

W niniejszym sylabusie opisano następujące techniki:
- testowanie instrukcji,
- testowanie decyzji,
- testowanie MC/DC,
- testowanie warunków wielokrotnych,
- testowanie API.

Sylabus Certyfikowany Tester – Poziom Podstawowy [CTFL_SYL] opisuje testowanie instrukcji i testowanie decyzji. Testowanie instrukcji skupia się na sprawdzaniu instrukcji wykonywalnych zawartych w kodzie, podczas gdy testowanie decyzji sprawdza wyniki decyzji.

Wymienione powyżej techniki opierają się na predykatach decyzyjnych zawierających wiele warunków i znajdują podobne typy defektów. Niezależnie od stopnia skomplikowania predykatu decyzji, będzie on mieć wartość PRAWDA albo FAŁSZ, co wyznacza określoną ścieżkę w kodzie. Defekt zostanie wykryty wówczas, gdy zakładana ścieżka nie będzie obsłużona ze względu na brak zgodności wartości predykatu decyzji z oczekiwaniami.

Więcej informacji na temat technik białoskrzynkowych można znaleźć w [ISO 29119] lub [Roman18].

2.2 Testowanie instrukcji

Testowanie instrukcji służy do sprawdzania instrukcji wykonywalnych zawartych w kodzie. Pokrycie mierzy się jako iloraz liczby instrukcji wykonanych przez testy przez łączną liczbę instrukcji wykonywalnych w przedmiocie testów. Pokrycie zwykle wyrażane jest w procentach.

Obszar zastosowania
Osiągnięcie pełnego pokrycia instrukcji powinno być traktowane jako minimalny wymóg w odniesieniu do całego testowanego kodu, choć w praktyce nie zawsze jest to możliwe.

Ograniczenia/trudności
Osiągnięcie pełnego pokrycia instrukcji powinno być traktowane jako minimalny wymóg w odniesieniu do całego testowanego kodu, choć w praktyce nie zawsze jest to możliwe ze względu na ograniczenia czasowe i/lub wielkość niezbędnego w tym celu wysiłku. Nawet wysoki wskaźnik pokrycia instrukcji kodu nie oznacza, że zostaną wykryte pewne defekty związane z logiką zawartą w kodzie. W wielu przypadkach osiągnięcie 100% pokrycia nie jest możliwe ze względu na nieosiągalny kod. Chociaż pisanie nieosiągalnego kodu ogólnie nie jest uważane za dobrą praktyką programowania, może się zdarzyć, że takie instrukcje znajdą się w kodzie, np. w postaci kodu dla domyślnego (ang. *default*) bloku „case” w instrukcji SWITCH-CASE,

w której wszystkie możliwe przypadki są poprawnie obsługiwane przez bloki „case” poprzedzające blok domyślny.

2.3 Testowanie decyzji

Testowanie decyzji służy do sprawdzania wyników decyzji zawartych w kodzie. W tym celu tworzy się przypadki testowe, które odzwierciedlają przepływy sterowania występujące po punkcie decyzyjnym (na przykład dla instrukcji IF istnieje jeden przepływ sterowania w przypadku spełnienia warunku (*true*) i jeden w przypadku jego niespełnienia (*false*); dla instrukcji SWITCH-CASE może istnieć wiele przepływów odpowiadających wszystkim możliwym wynikom; dla pętli LOOP istnieje przepływ, dla którego warunek pętli jest prawdziwy i przepływ, dla którego warunek pętli jest fałszywy.

Pokrycie mierzy się jako iloraz liczby wyników decyzji wykonanych przez testy przez łączną liczbę wyników decyzji w przedmiocie testów. Pokrycie zwykle wyrażane jest w procentach. Należy pamiętać, że pojedynczy przypadek testowy może sprawdzić wiele wyników decyzji.

W porównaniu z opisanymi poniżej technikami zmodyfikowanego testowania warunkowo-decyzyjnego i testowania warunków wielokrotnych, w testowaniu decyzji cała decyzja jest rozważana jako całość, a jej wynikiem jest zawsze pojedyncza wartość PRAWDA lub FAŁSZ, niezależnie od stopnia skomplikowania wewnętrznej struktury tej decyzji.

Testowanie gałęzi (ang. *branch testing*) to termin często używany zamiennie z „testowaniem decyzji”, ponieważ pokrycie wszystkich gałęzi i pokrycie wszystkich wyników decyzji może być osiągnięte przez te same testy. Testowanie gałęzi sprawdza gałęzie w kodzie. Gałąź rozumiana jest jako krawędź w grafie przepływu sterowania (bezpośrednie przejście od jednej instrukcji do innej). Dla programów nie zawierających decyzji podana powyżej definicja pokrycia decyzji przyjmuje postać ilorazu 0/0, co jest wartością niezdefiniowaną, niezależnie od tego, ile testów zostanie uruchomionych. Jednak w przypadku pokrycia gałęzi każdy test pokryje jedną sekwencję gałęzi prowadzącą od wejścia do wyjścia (zakładając pojedyncze wejście i pojedyncze wyjście), co będzie skutkować osiągnięciem 100% pokrycia gałęzi. Aby uzgodnić różnice w tych dwóch miarach pokrycia, w przypadku kodu nie zawierającego decyzji, standard ISO 29119-4 wymaga uruchomienia co najmniej jednego testu w celu osiągnięcia 100% pokrycia decyzji. W ten sposób 100% pokrycia decyzji i 100% pokrycia gałęzi są pojęciami tożsamymi dla prawie wszystkich programów. Wiele narzędzi testowych dostarczających mechanizmy pomiaru pokrycia, włączając w to narzędzia używane do testowania systemów krytycznych ze względów bezpieczeństwa, wykorzystuje podobne podejście.

Obszar zastosowania
Należy wziąć pod uwagę ten poziom pokrycia wówczas, gdy testowany kod jest ważny albo bardzo ważny, czyli gdy ma znaczenie krytyczne (patrz tabele w sekcji 2.8.2. dla systemów krytycznych ze względów bezpieczeństwa). Technika ta może być użyta zarówno w stosunku do kodu, jak i jakiegokolwiek modelu, który wykorzystuje punkty decyzyjne, np. modele procesu biznesowego.

Ograniczenia/trudności
W testowaniu decyzji nie uwzględnia się sposobu podejmowania decyzji z wieloma warunkami, dlatego defekty spowodowane pewnymi kombinacjami wyników warunków mogą pozostać niewykryte.

2.4 Testowanie MC/DC

W porównaniu z testowaniem decyzji, w którym brana jest pod uwagę cała decyzja i sprawdzane są jej wyniki PRAWDA i FAŁSZ, w testowaniu MC/DC (zmodyfikowanego pokrycia warunkowo-decyzyjnego, ang. *modified condition/decision coverage*, MC/DC) rozpatruje się sposób konstrukcji decyzji, która obejmuje wiele warunków atomowych (w przypadku, gdy decyzja składa się z pojedynczego warunku atomowego, technika sprowadza się do zwykłego testowania decyzji).

Każdy predykat decyzji składa się z jednego lub wielu warunków atomowych, z których każdy ma wartość logiczną (ang. *Boolean value*). Warunki są łączone logicznie w celu określenia wyniku całej decyzji. W tej technice sprawdza się niezależny i poprawny wpływ każdego warunku atomowego na wynik całej decyzji.

Technika ta zapewnia silniejszy poziom pokrycia niż pokrycie instrukcji i pokrycie decyzji, jeśli istnieją decyzje zawierające wiele warunków. Zakładając, że decyzja składa się z *N* unikalnych, parami niezależnych warunków atomowych, MC/DC dla pojedynczej decyzji wymaga zwykle wykonania tej decyzji *N*+1 razy. Testowanie MC/DC wymaga par testów, które wykazują, że zmiana wartości logicznej pojedynczego warunku atomowego może niezależnie od innych warunków atomowych wpłynąć na zmianę wyniku decyzji. Należy pamiętać, że pojedynczy przypadek testowy może spowodować wielokrotne wykonanie tej samej decyzji dla różnych kombinacji wartości jej warunków atomowych, dlatego pełne pokrycie MC/DC można czasami osiągnąć wykonując mniej niż *N*+1 osobnych przypadków testowych.

Obszar zastosowania
Ta technika jest stosowana w testach w przemyśle lotniczym, przemyśle samochodowym, a także w innych branżach, w systemach krytycznych ze względów bezpieczeństwa. Stosuje się ją w testach oprogramowania, którego ewentualna awaria może spowodować katastrofę. Testowanie MC/DC może być rozsądnym wyborem pomiędzy testowaniem decyzji a testowaniem warunków wielokrotnych (patrz podrozdział 2.5.), ze względu na liczbę koniecznych do przetestowania kombinacji warunków atomowych. Jest to kryterium silniejsze od testowania decyzji, ale wymaga przetestowania zdecydowanie mniejszej liczby warunków testowych w porównaniu z testowaniem warunków wielokrotnych, gdy decyzja składa się z wielu warunków atomowych.

Ograniczenia/trudności
Uzyskanie pokrycia MC/DC może okazać się skomplikowane, jeśli w decyzji z wieloma warunkami znajduje się wiele wystąpień tej samej zmiennej. W takiej sytuacji warunki mogą być ze sobą powiązane. W konkretnej decyzji zmiana wartości jednego warunku w taki sposób, aby jedynie z tego powodu zmienił się wynik całej decyzji, może okazać się niemożliwa. Jedną z metod rozwiązania tego problemu jest założenie, że wykorzystując technikę testowania MC/DC testuje się wyłącznie niepowiązane warunki atomowe. Inny sposób polega na indywidualnym analizowaniu każdego przypadku występowania decyzji, w której pojawiają się warunki powiązane.

Niektóre kompilatory i/lub interpretery zaprojektowano w taki sposób, aby podczas wyliczania wartości złożonego wyrażenia decyzyjnego w kodzie następowało tzw. zwarcie. Oznacza to, że w uruchomionym kodzie całe wyrażenie może nie być obliczane, jeśli końcowy wynik obliczeń można określić już po ustaleniu wartości części wyrażenia. Na przykład w trakcie obliczania wartości decyzji „A i B” nie ma powodu obliczać wartości warunku B, jeśli wartością warunku A jest FAŁSZ. Żadna z dostępnych wartości B nie jest w stanie zmienić rezultatu końcowego, zatem da się skrócić czas wykonywania kodu rezygnując z obliczania wartości tego warunku. Zwarcie (ang. *short-circuiting*) może mieć wpływ na zdolność do spełnienia pokrycia MC/DC, ponieważ niektóre wymagane testy mogą być niemożliwe do zrealizowania. Zazwyczaj istnieje możliwość konfiguracji kompilatora tak, by wyłączyć funkcję zwarcia na potrzeby testowania, ale ta metoda może nie być dopuszczalna w przypadku aplikacji krytycznych ze względów bezpieczeństwa, gdzie często wymaga się, aby kod testowany i kod produkcyjny były identyczne.

2.5 Testowanie warunków wielokrotnych

W rzadkich przypadkach może okazać się konieczne przetestowanie wszystkich możliwych kombinacji warunków atomowych, które mogą pojawić się w decyzji. Taki poziom testowania nazywany jest testowaniem warunków wielokrotnych. Zakładając *N* unikalnych, wzajemnie niezależnych warunków atomowych, pełne pokrycie warunków wielokrotnych dla decyzji może być osiągnięte poprzez wywołanie jej 2^N razy, dla wszystkich możliwych kombinacji wartości warunków atomowych. Należy pamiętać, że pojedynczy przypadek testowy może sprawdzać kilka kombinacji warunków, dlatego pełne pokrycie warunków wielokrotnych można czasami osiągnąć wykonując mniej niż 2^N osobnych przypadków testowych. Pokrycie mierzy się jako iloraz liczby przetestowanych kombinacji wartości warunków atomowych przez łączną liczbę takich kombinacji we wszystkich decyzjach w przedmiocie testów. Pokrycie zwykle wyrażane jest w procentach.

Obszar zastosowania
Ta technika jest stosowana do testowania oprogramowania wysokiego ryzyka oraz oprogramowania wbudowanego, które powinny działać w niezawodny, bezawaryjny sposób przez długi okres czasu.

Ograniczenia/trudności
Ponieważ liczba przypadków testowych wynika bezpośrednio z tablicy prawdy (ang. *truth table*) zawierającej wszystkie kombinacje wartości warunków atomowych, łatwo określić ten poziom pokrycia. Jednakże sama liczba przypadków testowych wymaganych w testowaniu warunków wielokrotnych może być jednak bardzo duża, dlatego w większości sytuacji właściwsze jest zastosowanie techniki testowania MC/DC.

Jeśli w używanym kompilatorze występuje mechanizm zwarcia, faktyczna liczba kombinacji wartości warunków atomowych do przetestowania zwykle ulega zmniejszeniu, w zależności od kolejności i zgrupowania operacji logicznych wykonywanych na warunkach atomowych.

2.6 Testowanie ścieżek podstawowych

Ten podrozdział został usunięty z sylabusa w wersji 4.0.

2.7 Testowanie API

Interfejs programowania aplikacji (ang. API – *Application Programming Interface*) to dobrze zdefiniowany interfejs umożliwiający komunikację z innym systemem, który dostarcza go wraz z określonymi usługami, takimi jak dostęp do zdalnego zasobu. Typowe usługi to: usługi sieciowe (ang. *web services*), korporacyjne magistrale usług (ang. *enterprise service buses - ESB*), bazy danych, komputery głównego szeregu (ang. *mainframes*) czy webowe interfejsy użytkownika (ang. *Web UIs*).

Testowanie API to w zasadzie rodzaj testowania, a nie konkretna technika. W pewnym zakresie testowanie API przypomina testowanie graficznego interfejsu użytkownika (ang. GUI – *graphical user interface*). Podejście to koncentruje się na analizowaniu wartości wejściowych i zwracanych danych.

Istotnym elementem badania API jest często testowanie negatywne. Programiści, którzy wykorzystują interfejsy API do uzyskiwania dostępu do usług zewnętrznych w stosunku do tworzonego przez nich kodu, mogą podejmować próby użycia interfejsów API w sposób niezgodny z ich przeznaczeniem. Oznacza to konieczność wprowadzenia odpornych mechanizmów obsługi błędów, zapobiegających niepoprawnemu działaniu. Może być niezbędne przeprowadzenie testowania kombinatorycznego wielu interfejsów, ponieważ interfejsy API są często używane wspólnie, a każdy z nich może zawierać różne parametry, których wartości da się połączyć na różne sposoby.

Interfejsy API są często luźno powiązane, co zwiększa prawdopodobieństwo wystąpienia utraconych transakcji i zakłóceń czasowych. Niezbędne jest zatem dokładne przetestowanie mechanizmów odtwarzania i ponawiania. Organizacja, która udostępnia interfejs API, musi zagwarantować bardzo wysoką dostępność wszystkich usług. Na ogół wymaga to rygorystycznego testowania niezawodności oraz zapewnienia obsługi infrastruktury przez wydawcę/dostawcę API.

Obszar zastosowania
Testowanie API jest szczególnie przydatne w przypadku systemów złożonych z podsystemów (systemy systemów), ponieważ pojawia się coraz więcej systemów rozproszonych i korzystających ze zdalnego przetwarzania w celu przekierowania części zadań na inne procesory. Przykłady takich zastosowań:

- wywołania systemu operacyjnego,
- architektury zorientowane na usługi (ang. SOA – *service-oriented architectures*),
- zdalne wywołania procedur (ang. RPC – *remote procedure calls*),
- usługi internetowe (*Web services*).

Efektem konteneryzacji oprogramowania jest podział programu na kilka kontenerów, które komunikują się ze sobą z wykorzystaniem mechanizmów takich jak wymienione na powyższej liście. Testowanie API powinno uwzględnić również takie interfejsy.

Ograniczenia/trudności
Bezpośrednie testowanie API na ogół wymaga zastosowania specjalistycznych narzędzi przez Technicznego Analityka Testów. Ponieważ zwykle nie istnieje interfejs graficzny bezpośrednio powiązany z interfejsem API, do skonfigurowania początkowego środowiska, serializacji danych, wywołania funkcji API i określenia wyniku wykorzystuje się odpowiednie narzędzia.

Pokrycie
Testowanie API to opis typu testów, a nie opis konkretnego poziomu pokrycia. Test interfejsu API powinien obejmować co najmniej wywołania funkcji API z realistycznymi, poprawnymi wartościami parametrów wejściowych oraz z wartościami niepoprawnymi w celu sprawdzenia obsługi wyjątków. W bardziej szczegółowych testach interfejsów API można przyjąć, że wszystkie wywoływalne obiekty powinny zostać sprawdzone co najmniej raz, albo że zrealizowano co najmniej raz każde możliwe wywołanie każdej funkcji API.

Reprezentacyjny transfer stanu (ang. *Representational State Transfer*, *REST*) jest stylem architektury oprogramowania. Wykorzystujące go usługi sieciowe (web serwisy), tzw. usługi RESTful, pozwalają na dostęp do zasobów sieciowych poprzez wykorzystanie jednolitego zbioru bezstanowych operacji. Istnieje szereg kryteriów pokrycia dla interfejsów API usług sieciowych RESTful [Web-7]. Można je podzielić na dwie grupy: kryteria pokrycia wejścia i kryteria pokrycia wyjścia. Kryteria wejścia mogą wymagać wykonania wszystkich możliwych operacji API, wykorzystanie wszystkich możliwych parametrów funkcji API, pokrycie sekwencji operacji (wywołań funkcji) API. Kryteria wyjścia mogą wymagać wygenerowania wszystkich możliwych kodów statusów wykonania operacji (poprawnych oraz błędnych) oraz wygenerowania odpowiedzi zawierających zasoby o wszystkich możliwych własnościach (lub typach własności).

Typy defektów
Typy defektów, które mogą zostać wykryte w trakcie testowania API, są dość odmienne. Powszechne są problemy z interfejsami. Są to między innymi problemy dotyczące obsługi danych, zależności czasowych, utraty transakcji i duplikowania transakcji, a także problemy z obsługą sytuacji wyjątkowych.

2.8 Wybór białoskrzynkowej techniki testowania

Wybór białoskrzynkowej techniki testowania zazwyczaj określony jest w terminach wymaganego poziomu pokrycia, które osiągane jest poprzez zastosowanie tej techniki. Na przykład, wymóg osiągnięcia 100% pokrycia instrukcji zazwyczaj prowadzi do wykorzystania techniki testowania instrukcji. Zazwyczaj najpierw stosuje się techniki czarnoskrzynkowe, a następnie mierzy się osiągnięte pokrycie i technika białoskrzynkowa jest używana tylko wtedy, gdy odpowiedni poziom pokrycia nie został osiągnięty. W niektórych przypadkach białoskrzynkowe techniki testowania mogą być użyte w mniej formalny sposób, aby zidentyfikować miejsca, w których można zwiększyć pokrycie (np. tworząc dodatkowe testy tam, gdzie poziomy pokrycia białoskrzynkowego są szczególnie niskie). W takich przypadkach nieformalnej miary pokrycia testowanie instrukcji jest zazwyczaj wystarczającą techniką.

Określając wymagany poziom pokrycia białoskrzynkowego dobrą praktyką jest definiowanie go jedynie na poziomie 100%. Powodem tego jest fakt, że niższe wartości pokrycia zazwyczaj są oznaką tego, iż część kodu nieprzetestowana przez uruchomione przypadki testowe to fragmenty najtrudniejsze do przetestowania. Te fragmenty są zazwyczaj najbardziej złożone i podatne na błędy. Zatem wymóg i faktyczne osiągnięcie np. 80% pokrycia może oznaczać, że kod zawierający dużą część wykrywalnych defektów jest nieprzetestowany. Z tego powodu białoskrzynkowe kryteria pokrycia opisane w standardach prawie zawsze wymagane są na poziomie 100%. Ścisłe definicje tych poziomów pokrycia mogą czasami sprawić, że osiągnięcie tych poziomów jest trudne lub wręcz niemożliwe. Jednakże kryteria opisane w normie ISO 29119-4 pozwalają na wyłączenie z obliczeń nieosiągalnych elementów pokrycia, dzięki czemu 100% pokrycia jest osiągalnym celem. Podczas określania wymaganego pokrycia białoskrzynkowego dla przedmiotu testów wystarczy uczynić to dla pojedynczego kryterium pokrycia (np. nie jest konieczne

jednoczesne wymaganie 100% pokrycia instrukcji i 100% pokrycia MC/DC). Rozważając kryteria pokrycia jedynie na poziomie 100% można utworzyć hierarchię kryteriów, w której jedne kryteria pokrycia subsumują inne. Kryterium pokrycia K1 subsumuje kryterium pokrycia K2, jeśli dla każdego modułu, systemu lub ich specyfikacji każdy zestaw przypadków testowych osiągający 100% pokrycia K1 osiąga również 100% pokrycia K2. Na przykład, pokrycie gałęzi subsumuje pokrycie instrukcji, ponieważ osiągnięcie 100% pokrycia gałęzi (przejść między instrukcjami) oznacza w szczególności, że testy musiały wykonać każdą instrukcję, a więc osiągnąć 100% pokrycia instrukcji. Dla białoskrzynkowych technik testowania opisanych w niniejszym sylabusie zachodzą następujące relacje: pokrycie decyzji i pokrycie gałęzi subsumują pokrycie instrukcji, pokrycie MC/DC subsumuje zarówno pokrycie decyzji, jak i pokrycie gałęzi, a pokrycie warunków wielokrotnych subsumuje pokrycie MC/DC (jeśli na poziomie 100% pokrycia utożsamimy ze sobą pokrycie gałęzi i decyzji, możemy powiedzieć, że subsumują się nawzajem).

Gdy określa się poziomy pokrycia białoskrzynkowego, które powinny być osiągnięte, często definiuje się różne poziomy dla różnych części systemu. Dzieje się tak dlatego, że poszczególne części systemu przyczyniają się w różnym stopniu do ryzyka. Na przykład w systemach lotniczych podsystemom związanym z systemami rozrywki dla pasażerów może zostać przypisany niższy poziom ryzyka niż podsystemowi kontroli lotu. Testowanie interfejsów jest wspólne dla wszystkich typów systemów i zazwyczaj jest wymagane dla wszystkich poziomów integralności dla systemów związanych z bezpieczeństwem (szczegóły dot. poziomów integralności opisano w sekcji 2.8.2.). Poziom wymaganego pokrycia dla testowania API będzie się zazwyczaj zwiększał wraz ze wzrostem powiązanego ryzyka (np. wyższy poziom ryzyka związanego z publicznym interfejsem może wymagać bardziej rygorystycznego testowania API).

Wybór właściwej białoskrzynkowej techniki testowania jest zwykle oparty na specyfice przedmiotu testów oraz związanych z nim postrzegalnych ryzyk. Jeśli przedmiot testów dotyczy systemów krytycznych ze względów bezpieczeństwa (tzn. gdy jego awaria może grozić utracie zdrowia, życia lub wyrządzeniem szkód środowiskowych), stosuje się odpowiednie standardy regulacyjne i definiuje precyzyjnie białoskrzynkowe poziomy pokrycia (zob. sekcja 2.8.2.). Jeśli z kolei przedmiot testów nie dotyczy systemów krytycznych ze względów bezpieczeństwa, wybór białoskrzynkowych poziomów pokrycia jest bardziej swobodny, ale wciąż powinien być oparty na postrzegalnym ryzyku, tak jak to opisano w sekcji 2.8.1.

2.8.1 Systemy niekrytyczne (niezwiązane z bezpieczeństwem)

Podczas wyboru białoskrzynkowej techniki testowania dla systemów niekrytycznych rozważa się zazwyczaj poniższe czynniki (ich kolejność jest przypadkowa):

- Kontrakt – jeśli kontrakt (umowa) wymaga osiągnięcia określonego poziomu pokrycia, nieosiągnięcie go jest potencjalnym naruszeniem warunków kontraktu.
- Klient – jeśli klient wymaga określonego poziomu pokrycia, np. jako część procesu planowania testów, nieosiągnięcie go może powodować problemy z klientem (np. konflikt).
- Standardy regulacyjne – w niektórych branżach (np. sektor finansowy) dla systemów o znaczeniu krytycznym dla powodzenia projektu stosuje się standardy regulacyjne, które definiują wymagane białoskrzynkowe kryteria pokrycia (zob. sekcja 2.8.2. w kwestii standardów regulacyjnych dla systemów krytycznych związanych z bezpieczeństwem).
- Strategia testów – jeśli strategia testów obowiązująca w organizacji określa wymagania dla białoskrzynkowego pokrycia kodu, to niedostosowanie się do strategii może grozić krytyką ze strony kierownictwa wyższego szczebla.
- Styl kodowania – jeśli kod pisany jest tak, by decyzje nie zawierały wielu warunków atomowych, wymaganie poziomu pokrycia takiego jak pokrycie MC/DC czy pokrycie warunków wielokrotnych byłoby stratą czasu.
- Dane historyczne o defektach – jeśli dane historyczne dotyczące efektywności w uzyskiwaniu określonego poziomu pokrycia sugerują, że byłoby właściwe wykorzystać to pokrycie dla danego przedmiotu testów, ignorowanie tego typu dostępnych danych może być ryzykowne. Tego typu dane historyczne mogą być dostępne w ramach projektu, organizacji lub całej branży (tzw. dane przemysłowe).
- Umiejętności i doświadczenie – jeśli dostępni testerzy, którzy mogą wykonać testy, nie są wystarczająco doświadczeni lub mają niskie umiejętności w stosowaniu określonej techniki

białoskrzynkowej, może ona być źle zrozumiana, co może skutkować wprowadzeniem niepotrzebnego ryzyka w przypadku wybrania tej techniki.
- Narzędzia – pokrycie białoskrzynkowe w praktyce może być mierzone jedynie za pomocą narzędzi do pomiaru pokrycia. Jeśli narzędzia mierzące określone pokrycie nie są dostępne, wymaganie pomiaru tego pokrycia może nieść ze sobą duże ryzyko.

Podczas wyboru białoskrzynkowej techniki testowania dla systemów niekrytycznych, Techniczny Analityk Testów ma więcej swobody w rekomendacji określonego pokrycia białoskrzynkowego, niż w przypadku systemów krytycznych ze względów bezpieczeństwa. Dokonane wybory są zazwyczaj kompromisem pomiędzy postrzeganym ryzykiem a kosztami, zasobami i czasem wymaganymi do łagodzenia tych ryzyk poprzez testy białoskrzynkowe. W niektórych przypadkach bardziej odpowiednie od testów białoskrzynkowych mogą okazać się inne podejścia do testowania lub stosowanie innych podejść do wytwarzania oprogramowania.

2.8.2 Systemy krytyczne ze względów bezpieczeństwa

Gdy testowane oprogramowanie jest częścią systemu krytycznego ze względów bezpieczeństwa, zazwyczaj stosuje się odpowiedni standard regulacyjny, który definiuje wymagane poziomy pokrycia. Standardy te zwykle wymagają przeprowadzenia analizy zagrożeń (ang. *hazard analysis*) dla systemu, a zidentyfikowane podczas tego procesu ryzyka są wykorzystane do przypisania poziomów nienaruszalności bezpieczeństwa (ang. *Safety Integrity Levels* - SIL) poszczególnym częściom systemu. Poziomy wymaganego pokrycia są definiowane dla każdego z poziomów nienaruszalności bezpieczeństwa.

Standard IEC 61508 („Bezpieczeństwo funkcjonalne elektrycznych/elektronicznych/programowalnych elektronicznych systemów związanych z bezpieczeństwem”) [IEC 61508] jest zbiorczym standardem wykorzystywanym w wyżej wymienionych celach. Teoretycznie może być on wykorzystany dla dowolnych systemów krytycznych ze względu na bezpieczeństwo, ale niektóre branże stworzyły jego własne, specyficzne warianty (np. ISO 26262 [ISO 26262] stosowany w przemyśle samochodowym), a inne – stworzyły swoje własne standardy (np. DO-178C [DO 178C] dla systemów lotniczych). Więcej informacji na temat ISO 26262 można znaleźć w sylabusie ISTQB Certyfikowany Tester – Tester Oprogramowania Motoryzacyjnego [CT_AuT_SYL].

Standard IEC 61508 definiuje cztery poziomy nienaruszalności bezpieczeństwa SIL (ang. *Safety Integrity Levels*, *SILs*). Każdy z nich zdefiniowany jest jako relatywny poziom redukcji ryzyka dostarczanej przez funkcję bezpieczeństwa, skorelowany z częstością (ang. *frequency*) i dotkliwością (ang. *severity*) postrzeganych zagrożeń. Gdy przedmiot testów wykonuje funkcję związaną z bezpieczeństwem, wyższe ryzyko awarii oznacza, że przedmiot testów powinien cechować się większą niezawodnością. Poniższa tabela pokazuje poziomy niezawodności związane z poszczególnymi poziomami nienaruszalności bezpieczeństwa SIL. Należy zauważyć, że poziom niezawodności SIL 4 dla operacji ciągłych jest ekstremalnie wysoki, ponieważ odpowiada on średniemu czasowi między awariami (ang. *Mean Time Between Failures*, *MTBF*) większemu niż 10.000 lat.

IEC 61508 SIL	Praca ciągła (prawdopodobieństwo niebezpiecznej awarii na godzinę)	Praca na żądanie (prawdopodobieństwo awarii dla żądania)
1	$\geq 10^{-6}$ do $<10^{-5}$	$\geq 10^{-2}$ do $<10^{-1}$
2	$\geq 10^{-7}$ do $<10^{-6}$	$\geq 10^{-3}$ do $<10^{-2}$
3	$\geq 10^{-8}$ do $<10^{-7}$	$\geq 10^{-4}$ do $<10^{-3}$
4	$\geq 10^{-9}$ do $<10^{-8}$	$\geq 10^{-5}$ do $<10^{-4}$

Rekomendacje dla białoskrzynkowych poziomów pokrycia związanych z poszczególnymi poziomami SIL pokazane są w poniższej tabeli. Wartość „wysoce rekomendowane” w praktyce oznacza, że osiągnięcie odpowiedniego poziomu pokrycia jest obowiązkowe. Z kolei wartość „rekomendowane” wielu praktyków rozumie w ten sposób, że osiągnięcie związanego z nią pokrycia jest jedynie opcjonalne i unika forsowania osiągnięcia go, podając ku temu odpowiednie uzasadnienie. Zatem na przykład przedmiot testów

przyporządkowany do poziomu SIL 3 jest zazwyczaj testowany tak, aby osiągnąć 100% pokrycia gałęzi (automatycznie powoduje to osiągnięcie 100% pokrycia instrukcji ze względu na subsumpcję).

Poziom SIL wg IEC 61508	100% pokrycia instrukcji	100% pokrycia gałęzi	100% pokrycia MC/DC
1	rekomendowane	rekomendowane	rekomendowane
2	wysoce rekomendowane	rekomendowane	rekomendowane
3	wysoce rekomendowane	wysoce rekomendowane	rekomendowane
4	wysoce rekomendowane	wysoce rekomendowane	wysoce rekomendowane

Warto zauważyć, że powyższe poziomy SIL oraz wymagania odnośnie poziomów pokrycia zdefiniowane w normie IEC 61508 są inne niż te zdefiniowane w normie ISO 26262, a te z kolei – inne niż w standardzie DO-178C.

3. Analiza statyczna i dynamiczna — 180 minut

Słowa kluczowe

anomalia, analiza dynamiczna, analiza przepływu danych, analiza przepływu sterowania, analiza statyczna, dziki wskaźnik, para definicja-użycie, wyciek pamięci, złożoność cyklomatyczna

Cele nauczania dotyczące analizy statycznej i dynamicznej

3.2 Analiza statyczna

TTA-3.2.1. (K3) Kandydat potrafi obliczyć złożoność cyklomatyczną i zastosować analizę przepływu sterowania w celu wykrycia ewentualnych anomalii w kodzie związanych z tym przepływem

TTA-3.2.2. (K3) Kandydat potrafi wykorzystać analizę przepływu danych w celu wykrycia ewentualnych anomalii w kodzie związanych z tym przepływem

TTA-3.2.3. (K3) Kandydat potrafi zaproponować sposoby zwiększenia utrzymywalności kodu za pomocą analizy statycznej

Uwaga: cel nauczania TTA-3.2.4. został usunięty z wersji 4.0 niniejszego sylabusa

3.3 Analiza dynamiczna

TTA-3.3.1. (K3) Kandydat potrafi zastosować analizę dynamiczną, aby osiągnąć określony cel

3.1 Wstęp

Analiza statyczna (zob. podrozdział 3.2.) jest rodzajem testowania przeprowadzanego bez uruchamiania oprogramowania. Jakość badanego oprogramowania jest oceniana przez człowieka lub przez narzędzie na podstawie jego budowy, struktury, zawartości lub dokumentacji. Statyczny obraz oprogramowania umożliwia przeprowadzenie szczegółowej analizy bez konieczności tworzenia danych i spełniania warunków wstępnych niezbędnych do wykonania przypadków testowych.

Poza analizą statyczną, do repertuaru metod testowania statycznego zalicza się również różne rodzaje przeglądów. Te z nich, które są istotne z punktu widzenia Technicznego Analityka Testów, zostały opisane w Rozdziale 5.

Analiza dynamiczna (zob. podrozdział 3.3.) wymaga faktycznego wykonania kodu i jest stosowana do wykrywania defektów, które łatwiej znaleźć, jeśli dany kod zostanie uruchomiony (np. wycieków pamięci). Podobnie jak w przypadku analizy statycznej, analiza dynamiczna może opierać się na zastosowaniu narzędzi lub monitorowaniu wykonywanego systemu przez człowieka pod kątem występowania takich objawów jak szybki przyrost używanej pamięci.

3.2 Analiza statyczna

Celem analizy statycznej jest wykrycie rzeczywistych i potencjalnych defektów w kodzie i architekturze systemu oraz poprawienie utrzymywalności kodu i architektury.

3.2.1 Analiza przepływu sterowania

Analiza przepływu sterowania to technika statyczna, w której kolejne kroki wykonywane w programie są analizowane przy użyciu grafu przepływu sterowania, zazwyczaj przy pomocy odpowiedniego narzędzia. Istnieją różne rodzaje anomalii, które można wykryć w systemie za pomocą tej techniki. To m.in. źle zaprojektowane pętle (np. pętle z wieloma punktami wejścia lub bez warunku stopu), niejednoznaczne obiekty docelowe wywołań funkcji w pewnych językach, niepoprawna kolejność wykonywanych operacji, nieosiągalny kod, niewywoływane funkcje itp.

Analizę przepływu sterowania można zastosować do ustalenia złożoności cyklomatycznej. Złożoność cyklomatyczna to dodatnia liczba całkowita określająca liczbę niezależnych ścieżek w silnie spójnym grafie.

Złożoność cyklomatyczna jest na ogół wykorzystywana jako wskaźnik złożoności komponentu. Według teorii przedstawionej przez Thomasa McCabe'a [McCabe76] im bardziej złożony jest system, tym więcej zawiera defektów i tym trudniejsze jest jego utrzymanie. W wielu pracach zauważono tego typu korelację między złożonością systemu a liczbą defektów. Każdy komponent, który odznacza się wysokim stopniem złożoności, powinien zostać poddany przeglądowi pod kątem możliwej refaktoryzacji, na przykład podzielenia go na kilka mniejszych jednostek.

3.2.2 Analiza przepływu danych

Analiza przepływu danych obejmuje wiele technik, które koncentrują się na gromadzeniu informacji o używaniu różnych zmiennych związanych z systemem. Wykorzystując graf przepływu sterowania bada się cykl życia każdej ze zmiennych, tj. miejsca jej zadeklarowania, zdefiniowania, użycia i usunięcia, ponieważ potencjalne anomalie mogą być wykryte w przypadku wykonania tych operacji w niewłaściwej kolejności [Beizer90].

Jedna z często używanych technik klasyfikuje każdą operację na zmiennej jako jedno z trzech atomowych działań:

- gdy zmienna jest definiowana, deklarowana lub inicjowana (np. x := 3),
- gdy zmienna jest używana lub odczytywana (np. if x > temp),
- gdy zmienna jest usuwana albo przestaje być dostępna w danym zasięgu (np. plik_tekstowy_1.zamknij, zmienna iterująca w pętli (i) po wyjściu z pętli itp.).

Sekwencje tych akcji, które mogą wskazywać na istnienie potencjalnych anomalii, to na przykład:

- definicja lub usuwanie następujące po innej definicji, bez uprzedniego użycia zmiennej,
- definicja bez następującego po niej usunięcia (może to prowadzić do wycieku pamięci w przypadku zmiennych alokowanych dynamicznie),
- użycie lub usunięcie przed definicją,
- użycie lub usunięcie po usunięciu.

Zależnie od języka programowania, niektóre z powyższych anomalii mogą zostać wykryte przez kompilator, ale czasem do identyfikacji anomalii w przepływie danych niezbędna może okazać się odrębna analiza statyczna wykonana przy pomocy narzędzia. Na przykład, ponowna definicja bez uprzedniego użycia zmiennej jest dopuszczalna w większości języków programowania i może być zastosowana w kodzie celowo, ale często będzie oznaczona przez narzędzie do analizy przepływu danych jako potencjalna anomalia, która powinna zostać sprawdzona.

Wykorzystanie ścieżek w grafie przepływu sterowania w celu określenia sekwencji operacji dla zmiennej może prowadzić do wykrycia potencjalnych anomalii, które nie mogą wystąpić w praktyce. Na przykład, narzędzia do analizy statycznej nie zawsze są w stanie zweryfikować, czy dana ścieżka jest wykonalna, ponieważ niektóre ścieżki są określane na podstawie wartości, jakie zmienne przyjmują w trakcie działania programu. Istnieje również klasa trudnych do identyfikacji przez narzędzia problemów dotyczących przepływu danych, w których analizowane dane są częścią struktur danych z dynamicznie przypisanymi zmiennymi, takich jak rekordy czy tablice. Narzędzia do analizy statycznej mają również trudności z identyfikowaniem potencjalnych anomalii w przepływie danych, gdy zmienne są dzielone pomiędzy równolegle działające wątki w programie, ponieważ sekwencja operacji na danych może być trudna do przewidzenia.

W odróżnieniu od analizy przepływu danych, która jest metodą testowania statycznego, testowanie przepływu danych jest formą testowania dynamicznego, w którym przypadki testowe generowane są tak, aby pokryć tzw. pary definicja-użycie w kodzie programu. Testowanie przepływu danych wykorzystuje te same koncepcje co analiza przepływu danych, ponieważ pary definicja-użycie są po prostu ścieżkami w przepływie sterowania, biegnącymi pomiędzy definicją a następującym po niej użyciem zmiennej w programie.

3.2.3 Analiza statyczna jako sposób poprawy utrzymywalności

Istnieją różne sposoby zastosowania analizy statycznej do poprawy utrzymywalności kodu, architektury i stron internetowych.

Źle napisany, nieskomentowany i nieustrukturyzowany kod jest trudniejszy w utrzymaniu. Odnalezienie i przeanalizowanie defektów w kodzie wymaga od programistów większego nakładu pracy, a modyfikacja kodu w celu naprawy defektu lub dodania nowej funkcji może skutkować wprowadzeniem kolejnych defektów.

Analiza statyczna pozwala na weryfikację zgodności ze standardami i wytycznymi dotyczącymi kodowania. Wykrycie niezgodności pozwala udoskonalić kod tak, by zwiększyć jego utrzymywalność. Te standardy i wytyczne opisują wymagane praktyki tworzenia kodu i projektowania, m.in. konwencje nazewnictwa, sposób komentowania, zasady tworzenia wcięć w kodzie i modularyzację. Należy zauważyć, że narzędzia do analizy statycznej zwykle zgłaszają ostrzeżenia, a nie wykrywają defekty. Ostrzeżenia (np. dotyczące stopnia złożoności kodu) mogą się pojawić nawet wówczas, gdy kod jest poprawny pod względem składniowym.

Modułowa konstrukcja systemu zwykle pozwala poprawić utrzymywalność kodu. Narzędzia do analizy statycznej wspierają tworzenie kodu modułowego w następujący sposób:

- Wyszukują powtarzający się kod. Takie fragmenty kodu potencjalnie nadają się do refaktoryzacji i przekształcenia w osobne funkcje (chociaż narzut związany z wywołaniami funkcji w czasie wykonania może być problemem w przypadku systemów czasu rzeczywistego).
- Generują metryki, które są przydatnymi wskaźnikami umożliwiającymi podział kodu na moduły. Mierzone są między innymi sprzężenie (stopień sprzężenia) (ang. *coupling*) i spójność (ang. *cohesion*) kodu. System odznaczający się dobrą utrzymywalnością na ogół ma niski wskaźnik sprzężenia (stopień powiązania modułów z innymi w czasie wykonywania) i wysoki wskaźnik spójności (stopnia, w jakim moduły są samodzielne i przeznaczone do realizacji pojedynczego zadania).
- Wskazują w kodzie obiektowym miejsca, gdzie obiekty pochodne mogą mieć zbyt dużą lub zbyt małą widoczność w klasie nadrzędnej.
- Wskazują obszary kodu lub architektury odznaczające się dużą złożonością strukturalną.

Narzędzia do analizy statycznej mogą również służyć do pielęgnowania stron internetowych. W takim wypadku należy sprawdzać, czy drzewo struktury strony jest zrównoważone i czy nie występują nieprawidłowości, które mogą prowadzić do:

- zwiększenia trudności zadań związanych z testowaniem,
- zwiększenia nakładu pracy związanego z pracami konserwacyjnymi.

Poza oceną utrzymywalności, narzędzia do analizy statycznej mogą być również zastosowane do kodu stron internetowych, aby sprawdzić ewentualne podatności na ataki zabezpieczeń, takie jak wstrzykiwanie kodu (ang. *code injection*), bezpieczeństwo ciasteczek (ang. *cookie security*), cross-site scripting, manipulowanie zasobami (ang. *resource tampering*) czy SQL injection. Więcej szczegółów na ten temat znajduje się w podrozdziale 4.3. oraz w sylabusie Certyfikowany Tester – Tester Zabezpieczeń [CT_SEC_SYL].

3.3 Analiza dynamiczna

3.3.1 Przegląd

Analiza dynamiczna służy do wykrywania awarii, których objawy są widoczne dopiero w trakcie wykonywania kodu. Przykładem takich awarii mogą być wycieki pamięci, które mogą być wykrywalne przez zastosowanie analizy statycznej (poprzez znalezienie kodu, który przydziela pamięć, ale nigdy jej nie zwalnia), natomiast dzięki analizie dynamicznej wycieki takie są łatwe do rozpoznania.

Awarie, których nie da się natychmiast odtworzyć (nieregularne), mogą mieć istotne konsekwencje dla pracochłonności procesu testowania oraz możliwości wprowadzenia oprogramowania do sprzedaży lub jego eksploatacji. Wśród przyczyn takich awarii można wymienić wycieki pamięci lub zasobów, nieprawidłowe stosowanie wskaźników oraz inne nieprawidłowości (np. dotyczące stosu systemowego) [Kaner02]. Takie awarie mogą powodować stopniowy spadek wydajności systemu, a nawet zaprzestanie jego działania. W strategiach testowania należy zatem uwzględnić ryzyka związane z tego rodzaju defektami i — w uzasadnionych przypadkach — przewidzieć również przeprowadzenie analizy dynamicznej mające na celu ograniczenie tego ryzyka (zwykle przy użyciu odpowiednich narzędzi). Powyższe awarie są często najkosztowniejsze do zlokalizowania i usunięcia, dlatego też zaleca się rozpoczęcie analizy dynamicznej na wczesnym etapie projektu.

Analizę dynamiczną można stosować w celu:

- zapobiegania wystąpieniu awarii poprzez wykrywanie wycieków pamięci (patrz sekcja 3.3.2.) i dzikich wskaźników (patrz sekcja3.3.3.),
- analizowania trudnych do odtworzenia awarii systemu,
- dokonywania oceny funkcjonowania sieci,
- zwiększania wydajności systemu dzięki wykorzystaniu profilerów kodu (ang. *code profilers*) w celu dostarczenia informacji o zachowaniu systemu w czasie wykonywania, które można następnie wykorzystać i wprowadzić odpowiednie zmiany.

Analizę dynamiczną można przeprowadzić na dowolnym poziomie testów. Wymaga ona jednak umiejętności technicznych i systemowych, które pozwalają:

- określić cele testów realizowane w ramach analizy dynamicznej,
- ustalić właściwy czas rozpoczęcia i zakończenia analizy,
- przeanalizować wyniki.

Narzędzia do analizy dynamicznej mogą być wykorzystywane nawet jeśli Techniczny Analityk Testów posiada minimalne kwalifikacje techniczne; narzędzia te tworzą zwykle obszerne dzienniki (logi testów), które mogą być następnie analizowane przez osoby posiadające niezbędne umiejętności techniczne i analityczne.

3.3.2 Wykrywanie wycieków pamięci

Z wyciekiem pamięci mamy do czynienia w sytuacji, w której obszary pamięci operacyjnej (RAM) są przydzielane programowi, ale nie są zwalniane, gdy przestają być potrzebne. Danego obszaru pamięci nie można wtedy ponownie wykorzystać. Jeśli dzieje się tak często lub jeśli dostępne zasoby pamięci są bardzo ograniczone, programowi może zabraknąć możliwej do wykorzystania pamięci. Dawniej odpowiedzialność za właściwe posługiwanie się pamięcią ponosił programista. Program dokonujący alokacji musiał zwalniać wszelkie dynamicznie przydzielane obszary pamięci w celu uniknięcia wycieku pamięci. Obecnie wiele środowisk programistycznych zawiera automatyczne lub półautomatyczne funkcje „odśmiecania" pamięci (ang. *garbage collection*), które pozwalają odzyskać alokowaną pamięć po jej użyciu bez bezpośredniej interwencji programisty. W tej sytuacji — gdy przydzielona pamięć powinna być zwalniana w ramach procesu automatycznego czyszczenia — wyizolowanie wycieków pamięci może być bardzo trudne.

Wycieki pamięci zazwyczaj powodują problemy dopiero po jakimś czasie – gdy wycieknie i przestanie być dostępna znacząca ilość pamięci. Wycieki pamięci nie są zauważalne, gdy oprogramowanie jest świeżo zainstalowane lub system był niedawno zresetowany, a pamięć została ponownie przydzielona; testowanie jest przykładem, w którym częste alokacje pamięci mogą zapobiec detekcji wycieków pamięci. Z tego powodu negatywne skutki wycieków pamięci są często dostrzegane dopiero po rozpoczęciu eksploatacji programu w środowisku produkcyjnym.

Głównym objawem wycieku pamięci jest stopniowe wydłużanie się czasu reakcji systemu, co może ostatecznie doprowadzić do jego awarii. Skutki takiej awarii można co prawda usunąć poprzez zrestartowanie (ponowne uruchomienie) systemu, ale jest to niewygodne, a niekiedy, w przypadku niektórych systemów, może nawet okazać się niemożliwe.

Wiele narzędzi do analizy dynamicznej pozwala zidentyfikować obszary kodu, w których występują wycieki pamięci, dzięki czemu można wprowadzić odpowiednie poprawki. Proste programy do monitorowania pamięci również pozwalają się zorientować, czy ilość dostępnej pamięci z czasem maleje, chociaż ustalenie dokładnej przyczyny spadku ilości pamięci wymaga w takim przypadku przeprowadzenia dalszej analizy.

3.3.3 Wykrywanie dzikich wskaźników

Dzikie wskaźniki w programie to wskaźniki, które przestały być poprawne i nie powinny być przez program używane. Dzikim wskaźnikiem może być na przykład wskaźnik, który „utracił" obiekt lub funkcję docelową, na którą wskazywał, bądź też wskaźnik, który odwołuje się do innego niż zamierzony obszaru pamięci (np. do obszaru znajdującego się poza przydzielonymi granicami tablicy). Użycie w programie dzikich wskaźników może mieć różne konsekwencje, na przykład:

- Program może działać zgodnie z oczekiwaniami. Sytuacja taka może mieć miejsce, jeśli dziki wskaźnik odwołuje się do pamięci, która nie jest obecnie używana przez program (przez co jest teoretycznie „wolna") i/lub zawiera rozsądną wartość niepowodującą problemów.
- Program może się zawiesić. Dzieje się tak, jeśli dziki wskaźnik spowoduje nieprawidłowe użycie obszaru pamięci mającego krytyczne znaczenie dla działania programu (np. obszaru zastrzeżonego dla systemu operacyjnego).

- Program może nie działać prawidłowo z powodu braku dostępu do wymaganych obiektów. W takiej sytuacji program może nadal funkcjonować, ale zostanie wyświetlony komunikat o błędzie.
- Wskaźnik może uszkodzić dane znajdujące się w określonym obszarze pamięci, czego skutkiem jest użycie niepoprawnych wartości (może to również stanowić zagrożenie zabezpieczeń).

Należy przy tym pamiętać, że zmiana związana z użyciem pamięci przez program (np. nowa wersja po wprowadzeniu modyfikacji w oprogramowaniu) może pociągnąć za sobą każdy z wyżej wymienionych skutków. Ma to szczególnie istotne znaczenie w sytuacji, w której program na początku działa zgodnie z oczekiwaniami pomimo użycia dzikich wskaźników, a następnie niespodziewanie zawiesza się (być może dopiero podczas eksploatacji) po wprowadzeniu zmiany. Narzędzia mogą pomóc w identyfikowaniu dzikich wskaźników używanych przez program niezależnie od tego, czy wpływają one na jego działanie. Niektóre systemy operacyjne zawierają wbudowane funkcje umożliwiające wykrywanie naruszeń zasad dostępu do pamięci w czasie wykonywania programu. System operacyjny może na przykład zgłosić wyjątek, gdy aplikacja próbuje uzyskać dostęp do miejsca w pamięci znajdującego się poza obszarem dozwolonym dla tej aplikacji.

3.3.4 Analiza wydajności

Analiza dynamiczna przydaje się nie tylko do wykrywania awarii i lokalizowania związanych z nimi defektów. Narzędzia używane podczas analizy dynamicznej wydajności programu pozwalają rozpoznać wąskie gardła związane z wydajnością oraz wygenerować wiele różnych metryk wydajności, które mogą posłużyć programistom do dostrojenia wydajności systemu. W ten sposób można na przykład uzyskać informacje o tym, ile razy moduł jest wywoływany w trakcie wykonywania programu, a następnie skupić się na zwiększaniu wydajności takich najczęściej wywoływanych modułów. Często działa tu tzw. zasada Pareto: program spędza większość swojego czasu wykonania (80%) w małej liczbie modułów (20%) [Andrist20].

Analizę dynamiczną wydajności programu często przeprowadza się w ramach testów systemu, ale można ją również wykonać podczas testowania pojedynczego podsystemu we wcześniejszych fazach testowania (z wykorzystaniem jarzm testowych). Dodatkowe informacje na ten temat można znaleźć w sylabusie Certyfikowany Tester – Tester Wydajności [CT_PT_SYL].

4. Charakterystyki jakościowe w testach technicznych — 345 minut

Słowa kluczowe
analizowalność, autentykacja, zachowanie w czasie, dojrzałość, osiągalność, instalowalność, integralność, charakterystyka jakościowa, model wzrostu niezawodności, modułowość, modyfikowalność, możliwość ponownego użycia, niezaprzeczalność, niezawodność, rozliczalność, odtwarzalność, pojemność, poufność, profil operacyjny, przenaszalność, testowalność, tolerowanie usterek, utrzymywalność, współistnienie, wydajność, zabezpieczenia, zastępowalność, zdolność adaptacyjna, kompatybilność, zużycie zasobów

Cele nauczania dotyczące charakterystyk jakościowych w testach technicznych

4.2. Zagadnienia dotyczące ogólnego planowania

TTA-4.2.1. (K4) Dla konkretnego scenariusza kandydat potrafi przeanalizować wymagania niefunkcjonalne i napisać odpowiednie fragmenty planu testów
TTA-4.2.2. (K3) Dla danego ryzyka produktowego kandydat potrafi określić konkretny typ(-y) testów niefunkcjonalnych, które są najbardziej odpowiednie
TTA-4.2.3. (K2) Kandydat rozumie i potrafi wyjaśnić etapy w cyklu życia oprogramowania, w których należy w typowych sytuacjach przeprowadzić testowanie niefunkcjonalne
TTA-4.2.4. (K3) Dla danego scenariusza kandydat potrafi zdefiniować typy defektów, których wykrycia należy się spodziewać w przypadku zastosowania różnego rodzaju typów testów niefunkcjonalnych

4.3. Testowanie zabezpieczeń

TTA-4.3.1. (K2) Kandydat potrafi wyjaśnić przyczyny uwzględnienia testowania zabezpieczeń w podejściu do testowania
TTA-4.3.2. (K2) Kandydat potrafi wyjaśnić główne aspekty, które należy uwzględnić podczas planowania i specyfikowania testów zabezpieczeń

4.4. Testowanie niezawodności

TTA-4.4.1. (K2) Kandydat potrafi wyjaśnić przyczyny uwzględnienia testowania niezawodności w podejściu do testowania
TTA-4.4.2. (K2) Kandydat potrafi wyjaśnić główne aspekty, które należy uwzględnić podczas planowania i specyfikowania testów niezawodności

4.5. Testowanie wydajnościowe

TTA-4.5.1. (K2) Kandydat potrafi wyjaśnić przyczyny uwzględnienia testowania wydajnościowego w podejściu do testowania
TTA-4.5.2. (K2) Kandydat potrafi wyjaśnić główne aspekty, które należy uwzględnić podczas planowania i specyfikowania testów wydajnościowych

4.6. Testowanie utrzymywalności

TTA-4.6.1. (K2) Kandydat potrafi wyjaśnić przyczyny uwzględnienia testowania utrzymywalności w podejściu do testowania

4.7. Testowanie przenaszalności

TTA-4.7.1. (K2) Kandydat potrafi wyjaśnić przyczyny uwzględnienia testowania przenaszalności w podejściu do testowania

4.8. Testowanie kompatybilności

TTA-4.8.1. (K2) Kandydat potrafi wyjaśnić przyczyny uwzględnienia testowania współistnienia w podejściu do testowania

4.1 Wstęp

Ogólnie rzecz biorąc, Techniczny Analityk Testów koncentruje się na testowaniu tego, "jak" działa produkt, a nie na aspektach funkcjonalnych ("co" produkt robi). Takie testy można przeprowadzić na dowolnym poziomie testów. Na przykład w trakcie testowania modułów systemów czasu rzeczywistego i systemów wbudowanych istotne jest przeprowadzenie testów porównawczych (ang. *benchmarking*) wydajności oraz testów zużycia zasobów. W trakcie produkcyjnych testów akceptacyjnych i testów systemowych właściwe jest testowanie aspektów związanych z niezawodnością, takich jak odtwarzalność. Testy na tym poziomie dotyczą konkretnego systemu, tj. kombinacji sprzętu i oprogramowania. Konkretny, testowany system, może zawierać różne serwery, klientów, bazy danych, sieci i inne zasoby. Niezależnie od poziomu testów, testowanie należy przeprowadzić z uwzględnieniem priorytetów ryzyka i dostępnych zasobów.

Zarówno testowanie dynamiczne, jak i testowanie statyczne, wliczając w to przeglądy (patrz Rozdziały 2, 3 oraz 5) może zostać zastosowane do testowania niefunkcjonalnych charakterystyk jakościowych opisanych w tym rozdziale.

Opis charakterystyk jakościowych produktu jest zawarty w normie ISO 25010 [ISO 25010] i służy jako przewodnik dla charakterystyk i ich podcharakterystyk. Zostały one przedstawione w poniższej tabeli wraz ze wskazaniem, które charakterystyki i podcharakterystyki są opisane w sylabusie dla analityków testów, a które w sylabusie dla technicznych analityków testów:

Charakterystyka	Podcharakterystyka	Analityk Testów	Techniczny Analityk Testów
Funkcjonalność (przydatność funkcjonalna)	poprawność funkcjonalna adekwatność funkcjonalna kompletność funkcjonalna	X	
Niezawodność	dojrzałość tolerowanie usterek odtwarzalność osiągalność		X
Użyteczność	stosowność łatwość nauki łatwość obsługi estetyka interfejsu użytkownika ochrona przed błędami użytkownika dostępność	X	
Wydajność	zachowanie w czasie zużycie zasobów pojemność		X
Utrzymywalność	analizowalność modyfikowalność testowalność modułowość możliwość ponownego użycia		X
Przenaszalność	zdolność adaptacyjna instalowalność zastępowalność	X	X
Zabezpieczenia	poufność integralność niezaprzeczalność rozliczalność autentykacja		X
Kompatybilność	współistnienie		X
	współdziałanie	X	

Załącznik A zawiera tabelę, w której zestawiono charakterystyki opisane w (obecnie wycofanym) standardzie ISO 9126-1 (używane w wersji 2012 niniejszego sylabusa) z charakterystykami opisanymi w późniejszym standardzie ISO 25010.

Aby można było sformułować i udokumentować odpowiednie podejście do testowania należy zidentyfikować czynniki ryzyka typowe dla wszystkich charakterystyk jakościowych i podcharakterystyk omówionych w tym podrozdziale. Testowanie charakterystyk jakościowych wymaga szczególnie starannego doboru właściwego momentu w cyklu życia, niezbędnych narzędzi, zastosowania wymaganych standardów, a także dostępności oprogramowania i dokumentacji oraz fachowej wiedzy technicznej. Bez zaplanowania podejścia dla każdej z charakterystyk i specyficznych potrzeb związanych z ich testowaniem tester może nie mieć do dyspozycji wystarczającej ilości czasu na odpowiednie zaplanowanie, przygotowanie i wykonanie testów.

Część tych testów, np. testowanie wydajnościowe, wymaga szczegółowego zaplanowania, udostępnienia specjalnego sprzętu i konkretnych narzędzi, specjalistycznych umiejętności dotyczących testowania oraz, w większości przypadków, dużej ilości czasu. Testowanie charakterystyk jakościowych i podcharakterystyk musi być powiązane z ogólnym harmonogramem testów i muszą być do niego przydzielone wystarczające zasoby.

Kierownik Testów zajmuje się kompilacją i raportowaniem informacji (w formie podsumowania) o metrykach dotyczących charakterystyk jakościowych i podcharakterystyk, natomiast Analityk Testów lub Techniczny Analityk Testów (zgodnie z powyższą tabelą) gromadzi informacje o każdej z metryk.

Pomiary charakterystyk jakościowych zebrane w testach przedprodukcyjnych przez Technicznego Analityka Testów mogą stanowić podstawę umów dotyczących poziomu świadczenia usług (umowy SLA – ang. *Service Level Agreement*) pomiędzy dostawcą systemu a interesariuszami (np. klientami lub operatorami). W niektórych sytuacjach testy mogą być kontynuowane po wdrożeniu produkcyjnym oprogramowania, przy czym często są wykonywane przez odrębny zespół lub organizację. Takie postępowanie ma na ogół miejsce w testach wydajności i niezawodności, których wyniki uzyskane w środowisku produkcyjnym mogą różnić się od wyników uzyskanych w środowisku testowym.

4.2 Zagadnienia dotyczące ogólnego planowania

Niezaplanowanie testowania niefunkcjonalnego stwarza poważne ryzyko niepowodzenia projektu. Kierownik Testów może poprosić Technicznego Analityka Testów o zidentyfikowanie głównych czynników ryzyka dla odpowiednich charakterystyk jakościowych (patrz tabela w podrozdziale 4.1.) i rozwiązanie wszelkich problemów dotyczących planowania, związanych z zaproponowanymi testami. Zagadnienia te można wykorzystać podczas opracowywania głównego planu testów.

Podczas wykonywania opisanych zadań należy uwzględnić następujące ogólne czynniki:

- wymagania interesariuszy,
- wymagania dotyczące środowiska testowego,
- zakup wymaganych narzędzi i szkolenia,
- kwestie organizacyjne,
- zagadnienia dotyczące bezpieczeństwa danych.

4.2.1 Wymagania interesariuszy

Wymagania niefunkcjonalne są często źle określone, a czasami ich brakuje – nie są w ogóle zdefiniowane. W fazie planowania Techniczny Analityk Testów musi być w stanie uzyskać od odpowiednich interesariuszy informacje o poziomach oczekiwań dotyczących charakterystyk jakościowych, a następnie dokonać oceny czynników ryzyka, które się z nimi łączą.

Zwykle przyjmuje się, że jeśli klient jest zadowolony z aktualnej wersji systemu, będzie także zadowolony z nowych wersji tak długo, jak długo będą utrzymywane osiągnięte poziomy jakości. Dzięki temu istniejąca wersja systemu może być traktowana jako punkt odniesienia (ang. *benchmark*). Podejście to jest szczególnie przydatne w przypadku niektórych niefunkcjonalnych charakterystyk jakościowych, takich jak wydajność, w przypadku których interesariusze mogą mieć problemy z określeniem wymagań.

W trakcie zbierania wymagań niefunkcjonalnych należy uwzględnić różne punkty widzenia. Wymagania powinny zostać pozyskane od przedstawicieli różnych grup, na przykład klientów i użytkowników, właścicieli produktów oraz personelu operacyjnego i serwisowego. Jeśli kluczowi interesariusze zostaną pominięci, z dużym prawdopodobieństwem niektóre wymagania zostaną pominięte. Więcej informacji o pozyskiwaniu wymagań można znaleźć w sylabusie Certyfikowany Tester – Poziom zaawansowany – Kierownik Testów [CT_TM_SYL].

W zwinnym wytwarzaniu oprogramowania wymagania niefunkcjonalne można zapisać w postaci historyjek użytkownika lub dodać do funkcjonalności określonej w przypadkach użycia jako ograniczenia niefunkcjonalne.

4.2.2 Wymagania dotyczące środowiska testowego

Wiele testów technicznych (np. testy zabezpieczeń, testy niezawodności i testy wydajnościowe) wymaga skonfigurowania środowiska testowego przypominającego środowisko produkcyjne w celu uzyskania realistycznych wartości pomiarów. W zależności od wielkości i złożoności testowanego systemu może mieć to istotny wpływ na planowanie i finansowanie testów. Ponieważ koszty takich środowisk mogą być wysokie, warto zastanowić się nad następującymi opcjami:

- wykorzystanie środowiska produkcyjnego,
- wykorzystanie ograniczonej wersji systemu, przy czym należy zadbać o to, by wyniki testów w wystarczającej mierze odzwierciedlały działanie systemu produkcyjnego,
- wykorzystanie zasobów w chmurze zamiast ich bezpośredniego zakupu,
- wykorzystanie środowisk zwirtualizowanych.

Ujęcie w harmonogramie momentu wykonywania takich testów powinno zostać starannie zaplanowane. Istnieje duże prawdopodobieństwo, że niektóre testy da się przeprowadzić jedynie w konkretnych terminach (np. w okresach zmniejszonego wykorzystania systemu).

4.2.3 Zakup wymaganych narzędzi i szkolenia

Narzędzia są częścią środowiska testowego. Komercyjne narzędzia lub symulatory są szczególnie istotne w przypadku testów wydajnościowych i niektórych rodzajów testów zabezpieczeń. Do zadań technicznych analityków testów należy oszacowanie kosztów i czasu potrzebnego na zakup, naukę i wdrożenie takich narzędzi. Jeśli mają zostać zastosowane specjalistyczne narzędzia, w planowaniu należy wziąć pod uwagę konieczność przyswojenia wiedzy na temat nowych narzędzi i koszt zaangażowania zewnętrznych specjalistów (w zakresie obsługi tych narzędzi).
Opracowanie złożonego symulatora może być odrębnym projektem rozwojowym i należy to uwzględnić w trakcie planowania. W szczególności w harmonogramie i planie wykorzystania zasobów należy przewidzieć testowanie i dokumentowanie opracowanego narzędzia. Na aktualizację i ponowne przetestowanie symulatora wynikające ze zmian w symulowanym produkcie trzeba przeznaczyć odpowiedni budżet i czas. W przypadku aplikacji krytycznych ze względów bezpieczeństwa plan prac nad symulatorami musi uwzględniać testowanie akceptacyjne i ewentualną certyfikację symulatora przez niezależną jednostkę.

4.2.4 Kwestie organizacyjne

Testy techniczne mogą wiązać się z pomiarami zachowania kilku modułów kompletnego systemu (np. serwerów, baz danych i sieci). Jeśli moduły są rozproszone między różnymi lokalizacjami i organizacjami, nakład pracy związany z zaplanowaniem i koordynacją testów może okazać się znaczny. Na przykład niektóre moduły oprogramowania mogą być dostępne na potrzeby testowania systemowego wyłącznie w określonych porach dnia. Może się także okazać, że organizacje są w stanie udzielić testerom wsparcia jedynie przez ograniczoną liczbę dni. Skutkiem braku potwierdzenia dostępności „na żądanie” modułów systemu i personelu pochodzącego z innych organizacji (np. "wypożyczonej" wiedzy specjalistycznej) mogą być poważne zakłócenia w przebiegu zaplanowanych testów.

4.2.5 Zagadnienia dotyczące bezpieczeństwa i ochrony danych

W fazie planowania testów muszą zostać wzięte pod uwagę szczególne środki bezpieczeństwa wdrożone w systemie, tak, aby możliwe było wykonanie wszystkich czynności związanych z testowaniem. Na przykład jeśli używane jest szyfrowanie danych, może ono utrudnić tworzenie danych testowych i weryfikację rezultatów testów.

Zasady i przepisy dotyczące ochrony danych mogą uniemożliwić generowanie wymaganych danych testowych opartych na danych produkcyjnych (np. danych osobowych i danych kart kredytowych). Anonimizacja danych testowych jest zadaniem nietrywialnym, które należy zaplanować w ramach implementacji testów.

4.3 Testowanie zabezpieczeń

4.3.1 Powody rozważenia testów zabezpieczeń

W ramach testowania zabezpieczeń dokonywana jest ocena podatności systemu na zagrożenia poprzez podjęcie próby naruszenia polityki bezpieczeństwa systemu. Poniżej znajduje się lista potencjalnych zagrożeń, które należy zbadać podczas testowania zabezpieczeń:

- Kopiowanie aplikacji lub danych bez upoważnienia.
- Nieautoryzowana kontrola dostępu (np. możliwość wykonywania zadań, do których użytkownik nie ma uprawnień). Prawa użytkowników, zasady dostępu i uprawnienia są przedmiotem tych testów. Takie informacje powinny być dostępne w specyfikacji systemu.
- Oprogramowanie, które wykazuje niezamierzone skutki uboczne podczas wykonywania zamierzonej (zaplanowanej) funkcji. Na przykład odtwarzacz multimedialny, który poprawnie odtwarza pliki audio, ale zapisuje przy tym pliki w nieszyfrowanej pamięci tymczasowej, wykazuje efekt uboczny, który może zostać wykorzystany przez cyberprzestępców.
- Kod umieszczony na stronie internetowej, który może zostać uruchomiony przez kolejnych użytkowników (ang. *cross-site scripting*, XSS). Kod taki może być złośliwy (ang. *malicious*).
- Przepełnienie bufora (ang. *buffer overrun*, *buffer overflow*), które może być spowodowane wprowadzaniem w polu wejściowym w interfejsie użytkownika łańcuchów o długości większej niż możliwa do poprawnego obsłużenia w kodzie. Luka w zabezpieczeniach przepełnienia bufora stanowi okazję do uruchomienia złośliwego kodu.
- Odmowa usługi, która uniemożliwia użytkownikom interakcję z aplikacją (przyczyną może być np. przeciążenie serwera WWW ustawicznie ponawianymi żądaniami).
- Przechwycenie, naśladowanie lub modyfikowanie, a następnie przekierowanie informacji (np. transakcji dokonywanych kartą kredytową) przez stronę trzecią w taki sposób, że użytkownik pozostaje nieświadomy istnienia strony trzeciej (tzw. atak „człowiek pośrodku”, ang. *man-in-the-middle attack*).
- Złamanie szyfrów używanych do ochrony wrażliwych danych.
- Bomby logiczne (nazywane czasami „kukułczymi jajami”), które mogą zostać rozmyślnie umieszczone w kodzie i aktywowane jedynie w pewnych okolicznościach (np. konkretnego dnia). Po aktywowaniu bomba logiczna może wykonać szkodliwe działania, takie jak usunięcie plików albo sformatowanie dysków.

4.3.2 Planowanie testów zabezpieczeń

Ogólnie rzecz biorąc, podczas planowania testów zabezpieczeń należy w szczególności zastanowić się nad następującymi zagadnieniami:

- Ponieważ problemy dotyczące zabezpieczeń mogą się pojawić w trakcie tworzenia architektury, projektowania i implementacji systemu, można zaplanować testy zabezpieczeń na poziomie testowania modułowego, integracyjnego i systemowego. Ze względu na zmieniający się charakter zagrożeń bezpieczeństwa testy zabezpieczeń można także wykonywać regularnie po wdrożeniu produkcyjnym systemu. Jest to szczególnie istotne w przypadku dynamicznej, otwartej architektury, takiej jak Internet rzeczy (ang. *Internet of Things* - IoT), w której w fazie produkcyjnej pojawia się wiele aktualizacji używanego oprogramowania i elementów sprzętowych.

- Podejścia do testowania zaproponowane przez Technicznego Analityka Testów mogą uwzględniać przeglądy architektury, projektu i kodu oraz analizę statyczną kodu z wykorzystaniem narzędzi do testowania zabezpieczeń. Narzędzia tego typu mogą być skutecznym środkiem wykrywania problemów związanych z zabezpieczeniami, które łatwo można pominąć w trakcie testowania dynamicznego.
- Techniczny Analityk Testów może być poproszony o zaprojektowanie i wykonanie pewnych typów „ataków" (patrz poniżej), które wymagają szczegółowego zaplanowania i skoordynowania działań z interesariuszami (w tym specjalistami w dziedzinie testowania zabezpieczeń). Inne testy zabezpieczeń można wykonać we współpracy z programistami lub analitykami testów (np. testowanie praw użytkowników, zasad dostępu i uprawnień).
- Kluczowym aspektem planowania testów zabezpieczeń jest uzyskanie akceptacji działań. Techniczny Analityk Testów musi uzyskać formalne zezwolenie od Kierownika Testów na wykonanie zaplanowanych testów zabezpieczeń. Wszelkie dodatkowe, niezaplanowane testy, mogą zostać uznane za rzeczywiste ataki, a osoba wykonująca takie testy jest narażona na podjęcie wobec niej działań prawnych. W przypadku braku potwierdzenia planów i uzyskania akceptacji w formie pisemnej wytłumaczenie typu „my tylko prowadziliśmy testy zabezpieczeń" może nie zabrzmieć zbyt przekonująco.
- Planowanie testów zabezpieczeń należy skoordynować z Dyrektorem ds. Bezpieczeństwa Informacji, jeśli w danym przedsiębiorstwie lub instytucji istnieje takie stanowisko.
- Należy zwrócić uwagę, że udoskonalenia, których celem jest poprawa zabezpieczeń systemu, mogą negatywnie wpłynąć na jego wydajność lub niezawodność. Po wprowadzeniu takich modyfikacji zalecane jest rozważenie konieczności wykonania testów wydajnościowych lub niezawodności (patrz podrozdziały 4.4. i 4.5.).

Podczas planowania testów zabezpieczeń konieczne może być uwzględnienie określonych standardów, takich jak [IEC 62443-3-2], który ma zastosowanie do systemów automatyki przemysłowej i systemów sterowania.

Sylabus Certyfikowany Tester – Tester Zabezpieczeń [CT_SEC_SYL] zawiera więcej informacji na temat głównych elementów planu testów zabezpieczeń.

4.3.3 Specyfikacja testów zabezpieczeń

Testy zabezpieczeń można pogrupować (zgodnie z pracą [Whittaker04]) według pochodzenia ryzyka zabezpieczeń. Obejmuje to między innymi zagrożenia:

- Związane z interfejsem użytkownika — nieautoryzowany dostęp i dane wejściowe wywołujące szkodliwe skutki.
- Związane z systemem plików — dostęp do wrażliwych danych przechowywanych w plikach lub repozytoriach.
- Związane z systemem operacyjnym — przechowywanie poufnych informacji (np. haseł) w postaci niezaszyfrowanej w pamięci, której zawartość może zostać ujawniona, gdy system ulegnie awarii wywołanej przez szkodliwe dane wejściowe.
- Związane z zewnętrznym oprogramowaniem — interakcje, które mogą wystąpić między zewnętrznymi modułami używanymi przez system. Takie problemy mogą pojawić się na poziomie sieci (np. przesłanie niepoprawnych pakietów lub komunikatów) lub na poziomie modułów oprogramowania (np. awaria modułu oprogramowania niezbędnego do działania systemu).

Podcharakterystyki zabezpieczeń zdefiniowane w standardzie ISO 25010 [ISO 25010] również mogą być podstawą specyfikacji testów. Obejmują one następujące aspekty zabezpieczeń:

- poufność,
- integralność,
- niezaprzeczalność,
- rozliczalność,
- autentykację.

Podczas projektowania testów zabezpieczeń można skorzystać z następującego podejścia [Whittaker04]:

- Zbierz informacje, które mogą okazać się przydatne podczas specyfikowania testów, np. nazwiska pracowników, adresy fizyczne, szczegółowe informacje o sieciach wewnętrznych, adresy IP, sygnatury używanego oprogramowania i sprzętu oraz wersja systemu operacyjnego.
- Wykonaj skanowanie podatności za pomocą ogólnie dostępnych narzędzi. Narzędzia tego rodzaju nie służą bezpośrednio do złamania systemu(-ów), ale do zidentyfikowania podatności zabezpieczeń, które stanowią naruszenie zasad ochrony lub mogą takie naruszenie spowodować. Określone podatności da się także zidentyfikować z wykorzystaniem dodatkowych informacji i list kontrolnych, takich jak listy opublikowane przez National Institute of Standards and Technology (NIST) [Web-1] oraz Open Web Application Security Project™ (OWASP) [Web-4].
- Korzystając ze zgromadzonych informacji, opracuj „plany ataków" (tj. plany działań testowych zmierzających do naruszenia zasad ochrony konkretnego systemu). W planach ataków trzeba wyspecyfikować wykorzystanie różnych danych wejściowych wprowadzanych za pośrednictwem różnych interfejsów (np. interfejsu użytkownika i systemu plików), tak, aby możliwe było wykrycie najpoważniejszych defektów związanych z zabezpieczeniami. Różne „ataki" opisane w pracy [Whittaker04] są cennym źródłem technik opracowanych specjalnie w celu testowania zabezpieczeń.

Plany ataków można opracować na potrzeby testowania penetracyjnego.

Dodatkowe informacje na temat testowania zabezpieczeń można znaleźć w podrozdziale 3.2. (poświęconym analizie statycznej) i w sylabusie Certyfikowany Tester – Tester Zabezpieczeń [CT_SEC_SYL].

4.4 Testowanie niezawodności

4.4.1 Wstęp

Klasyfikacja ISO 25010 charakterystyk jakościowych produktów definiuje następujące podcharakterystyki niezawodności: dojrzałość, osiągalność, tolerowanie usterek i odtwarzalność. Testowanie niezawodności dotyczy zdolności systemu lub oprogramowania do wykonywania określonych funkcji w określonych warunkach przez określony czas.

4.4.2 Testowanie dojrzałości

Dojrzałość to stopień, w jakim system (lub oprogramowanie) spełnia wymagania dotyczące niezawodności w normalnych warunkach eksploatacji, które zwykle określane są za pomocą profilu operacyjnego (patrz podrozdział 4.9.). Miary dojrzałości, jeśli są stosowane, często stanowią jedno z kryteriów wydania systemu (związane z udostępnieniem wersji produkcyjnej).

Tradycyjnie dojrzałość jest określana i mierzona dla systemów o wysokiej niezawodności, takich jak te związane z funkcjami krytycznymi dla bezpieczeństwa (np. system kontroli lotu samolotu), gdzie cel dojrzałości jest zdefiniowany jako część normy regulacyjnej. Wymogiem dojrzałości dla takiego systemu o wysokiej niezawodności może być średni czas między awariami (ang. *Mean Time Between Failures* - MTBF) do 10^9 godzin (chociaż jest to praktycznie niemożliwe do zmierzenia).

Zwykłe podejście do testowania dojrzałości systemów o wysokiej niezawodności znane jest jako modelowanie wzrostu niezawodności i zwykle ma miejsce pod koniec testowania systemu, po zakończeniu testowania innych charakterystyk jakościowych i usunięciu wszelkich usterek związanych z wykrytymi awariami. Jest to podejście statystyczne, zwykle wykonywane w środowisku testowym jak najbardziej zbliżonym do środowiska operacyjnego. Aby zmierzyć określony MTBF, generowane są dane wejściowe do testów w oparciu o profil operacyjny, a system jest uruchamiany i rejestrowane (a następnie usuwane) są awarie. Zmniejszająca się częstotliwość awarii pozwala przewidzieć MTBF przy użyciu modelu wzrostu niezawodności.

Jeśli dojrzałość jest celem dla systemów o niższej niezawodności (np. niezwiązanych z bezpieczeństwem), można wykorzystać liczbę awarii zaobserwowanych podczas określonego okresu spodziewanego

użytkowania operacyjnego (np. nie więcej niż dwie awarie o dużym wpływie na tydzień) i można ją zapisać jako część umowy dotyczącej poziomu świadczenia usług (SLA) dla systemu.

4.4.3 Testowanie osiągalności

Osiągalność jest zazwyczaj określana w kategoriach czasu, w którym system (lub oprogramowanie) jest dostępny dla użytkowników i innych systemów w normalnych warunkach pracy. Systemy mogą mieć niską dojrzałość, ale nadal mieć wysoką osiągalność (dostępność). Na przykład sieć telefoniczna może nie być w stanie połączyć kilku połączeń (a zatem mieć niską dojrzałość), ale tak długo, jak system przywracany jest szybko i pozwala na kolejne próby połączenia, większość użytkowników będzie zadowolona. Jednakże pojedyncza awaria powodująca przerwę w działaniu sieci telefonicznej na kilka godzin stanowiłaby niedopuszczalny poziom osiągalności. Osiągalność (dostępność) jest często określana jako część umowy SLA i mierzona dla systemów eksploatacyjnych (ang. *operational systems*), takich jak strony internetowe i aplikacje typu *software as a service* (SaaS). Dostępność systemu może być opisana jako 99,999% ("pięć dziewiątek"), co oznacza, że system może być niedostępny nie więcej niż przez 5 minut rocznie; alternatywnie osiągalność (dostępność) systemu może być określona w kategoriach niedostępności (np. system nie powinien być niedostępny dłużej niż przez 60 minut miesięcznie).

Pomiar osiągalności (dostępności) przed rozpoczęciem użytkowania (np. w ramach podejmowania decyzji o dopuszczeniu do produkcji) jest często wykonywany przy użyciu tych samych testów, które służą do pomiaru dojrzałości; testy są oparte na profilu operacyjnym oczekiwanego użytkowania przez dłuższy czas i wykonywane są w środowisku testowym jak najbardziej zbliżonym do środowiska operacyjnego. Osiągalność (dostępność) można zmierzyć jako MTTF / (MTTF + MTTR), gdzie MTTF (ang. *Mean Time To Failure*) to średni czas do wystąpienia awarii, a MTTR (ang. *Mean Time To Repair*) to średni czas do naprawy, który często mierzy się w ramach testów utrzymywalności. Jeżeli system charakteryzuje się wysoką niezawodnością i posiada zdolność do odzyskiwania danych (patrz sekcja 4.4.5.), wówczas w równaniu można zastąpić MTTR średnim czasem do odzyskania danych, jeżeli system potrzebuje pewnego czasu na odzyskanie danych po awarii.

4.4.4 Testowanie tolerowania usterek

Systemy (lub oprogramowanie) o bardzo wysokich wymaganiach w zakresie niezawodności często posiadają konstrukcję tolerującą usterki, co w idealnym przypadku pozwala na kontynuację działania systemu bez zauważalnych przestojów w przypadku wystąpienia awarii. Główną miarą odporności systemu na usterki jest jego zdolność do tolerowania awarii. Testowanie odporności na usterki obejmuje zatem symulację awarii w celu określenia, czy system może kontynuować działanie w przypadku wystąpienia takiej awarii. Identyfikacja potencjalnych warunków awarii do przetestowania jest ważną częścią testowania tolerancji na awarie.

Projekt odporny na usterki będzie zazwyczaj obejmował jeden lub więcej zduplikowanych podsystemów, zapewniając w ten sposób pewien poziom redundancji w przypadku awarii. W przypadku oprogramowania takie zduplikowane systemy muszą być rozwijane niezależnie, aby uniknąć typowych przyczyn awarii (ang. *common mode failures*); podejście to znane jest jako programowanie N-wersji (ang. *N-version programming*). Systemy sterowania lotem statku powietrznego mogą obejmować trzy lub cztery poziomy redundancji, przy czym najbardziej krytyczne funkcje są realizowane w kilku wariantach. Gdy problemem jest niezawodność sprzętu, system wbudowany może działać na wielu różnych procesorach, podczas gdy krytyczna strona internetowa może działać z serwerem lustrzanym (awaryjnym) wykonującym te same funkcje, który jest zawsze dostępny, aby przejąć kontrolę w przypadku awarii serwera głównego. Niezależnie od tego, jakie podejście do odporności na usterki jest zaimplementowane, testowanie jej zazwyczaj wymaga zarówno wykrycia usterki, jak i późniejszej reakcji na usterkę, która ma zostać przetestowana.

Testy wstrzykiwania błędów umożliwiają testowanie odporności systemu w obecności usterek w środowisku systemu (np. wadliwe zasilanie, źle sformułowane komunikaty wejściowe, niedostępny proces lub usługa, nieznaleziony plik lub niedostępna pamięć) oraz usterek w samym systemie (np. odwrócony bit spowodowany promieniowaniem kosmicznym, słaby projekt lub złe kodowanie). Testy wstrzykiwania

błędów są formą testów negatywnych - celowo wstrzykujemy defekty do systemu, aby upewnić się, że reaguje on w sposób, którego oczekiwaliśmy (np. bezpiecznie dla systemu związanego z bezpieczeństwem). Czasami scenariusze defektów, które testujemy, nie powinny nigdy wystąpić (np. zadanie oprogramowania nie powinno nigdy "umrzeć" lub utknąć w nieskończonej pętli) i nie mogą być symulowane przez tradycyjne testowanie systemu, ale z wykorzystaniem testowania wstrzykiwania błędów tworzymy scenariusz defektu i mierzymy późniejsze zachowanie systemu, aby upewnić się, że wykrywa on i radzi sobie z awarią.

4.4.5 Testowanie odtwarzalności

Odtwarzalność jest miarą zdolności systemu (lub oprogramowania) do odzyskania danych po awarii, zarówno pod względem czasu potrzebnego do odzyskania danych (który może być zredukowany do zmniejszonego stanu działania), jak i ilości utraconych danych. Podejścia do testowania odtwarzalności obejmują testowanie przejścia na zasilanie awaryjne oraz testowanie tworzenia kopii zapasowych i ich przywracania; oba te podejścia zazwyczaj obejmują procedury testowania oparte na przebiegach próbnych ("na sucho" - ang. *dry runs*) i tylko sporadyczne, a najlepiej niezapowiedziane, testy praktyczne w środowiskach produkcyjnych.

Testowanie tworzenia i odtwarzania kopii zapasowych koncentruje się na testowaniu procedur minimalizujących wpływ awarii na dane systemowe. Testy sprawdzają procedury zarówno tworzenia kopii zapasowych, jak i odtwarzania danych. Podczas gdy testowanie tworzenia kopii zapasowych danych jest stosunkowo proste, testowanie przywracania systemu z kopii zapasowej może być bardziej złożone i często wymaga starannego planowania, aby zapewnić minimalizację zakłóceń w działaniu systemu operacyjnego. Miary obejmują czas potrzebny do wykonania różnych typów kopii zapasowych (np. pełnej i przyrostowej), czas potrzebny do odtworzenia danych (docelowy czas odtworzenia) oraz poziom utraty danych, który jest akceptowalny (docelowy punkt odtworzenia).

Testowanie pracy mimo awarii jest wykonywane, gdy architektura systemu składa się zarówno z systemu podstawowego, jak i awaryjnego, który przejmie zadania w przypadku awarii systemu podstawowego. W przypadku, gdy system musi być w stanie odzyskać dane po wystąpieniu bardzo poważnej awarii (np. powodzi, ataku terrorystycznym lub poważnym ataku typu *ransomware*), testowanie pracy mimo awarii jest często nazywane testowaniem odzyskiwania danych po awarii, a system (lub systemy) awaryjny może często znajdować się w innej lokalizacji geograficznej. Przeprowadzenie pełnego testu odzyskiwania danych po awarii na systemie produkcyjnym wymaga niezwykle starannego planowania ze względu na ryzyko i utrudnienia (często związane z wolnym czasem kierowników wyższego szczebla, który może być poświęcony na zarządzanie odzyskiwaniem danych). Jeśli pełny test odzyskiwania danych po awarii zakończy się niepowodzeniem, natychmiast powracamy do systemu podstawowego (ponieważ tak naprawdę nie został on zniszczony!). Testy pracy mimo awarii obejmują testowanie przejścia na system awaryjny, a po przejęciu przezeń kontroli – weryfikację zapewnienia wymaganego poziomu usług.

4.4.6 Planowanie testów niezawodności

Podczas planowania testów niezawodności należy w szczególności zastanowić się nad następującymi zagadnieniami:

- Terminy. Testy niezawodności wymagają zazwyczaj przetestowania kompletnego systemu oraz wcześniejszego ukończenia innych rodzajów testów, a ich przeprowadzenie może zająć dużo czasu.
- Koszty. Systemy o wysokiej niezawodności są z reguły kosztowne w testowaniu z powodu długich okresów, w których systemy muszą być testowane bez awarii, aby można było przewidzieć wymagany wysoki MTBF.
- Czas trwania. Testowanie dojrzałości z wykorzystaniem modeli wzrostu niezawodności opiera się na wykrytych awariach, a w przypadku wysokich poziomów niezawodności uzyskanie statystycznie istotnych wyników zajmie dużo czasu.
- Środowisko testowe. Środowisko testowe musi być jak najbardziej zbliżone do środowiska produkcyjnego. Można wykorzystać też samo środowisko produkcyjne, ale może to być uciążliwe

dla użytkowników i może wiązać się z wysokim ryzykiem, jeśli na przykład test przywracania systemu po awarii wpłynie negatywnie na system produkcyjny.
- Zakres. Różne podsystemy i komponenty mogą być testowane pod kątem różnych rodzajów i poziomów niezawodności.
- Kryteria wyjścia. Wymagania dotyczące niezawodności powinny być określone przez standardy regulacyjne dla aplikacji związanych z bezpieczeństwem.
- Awaria. Pomiary niezawodności są w dużym stopniu uzależnione od zliczania awarii, dlatego należy z góry uzgodnić, co rozumie się przez awarię.
- Programiści. W przypadku testowania dojrzałości z wykorzystaniem modeli wzrostu niezawodności należy osiągnąć porozumienie z programistami, że zidentyfikowane usterki będą usuwane tak szybko, jak to możliwe.
- Pomiar niezawodności operacyjnej jest stosunkowo prosty w porównaniu z pomiarem niezawodności przed wydaniem, ponieważ musimy mierzyć tylko awarie; może to wymagać współpracy z zespołem operacyjnym.
- Wczesne testowanie. Osiągnięcie wysokiej niezawodności (w przeciwieństwie do jej pomiaru) wymaga jak najwcześniejszego rozpoczęcia testowania, z rygorystycznymi przeglądami wczesnych dokumentów bazowych i statyczną analizą kodu.

4.4.7 Specyfikacja testów niezawodności

W przypadku testowania dojrzałości i dostępności testowanie opiera się w dużej mierze na testowaniu systemu w normalnych warunkach eksploatacji. W przypadku takich testów wymagany jest profil operacyjny, który określa, w jaki sposób system ma być używany. Więcej szczegółów na temat profili operacyjnych znajduje się w podrozdziale 4.9.

W przypadku testowania odporności na usterki i możliwości odzyskania danych często konieczne jest wygenerowanie testów, które powielają awarie w środowisku i w samym systemie, aby określić reakcję systemu. Do tego celu często wykorzystuje się testy wstrzykiwania błędów (ang. *fault injection testing*). Dostępne są różne techniki i listy kontrolne służące do identyfikacji możliwych usterek i odpowiadających im awarii (np. analiza drzewa usterek, analiza trybu i skutków awarii).

4.5 Testowanie wydajnościowe

4.5.1 Wprowadzenie

Klasyfikacja cech jakościowych produktu opisana w normie ISO 25010 definiuje następujące podcharakterystyki wydajności: zachowanie w czasie, zużycie zasobów i przepustowość. Testowanie wydajności (związane z charakterystyką jakościową „wydajność”) dotyczy pomiaru wydajności systemu lub oprogramowania w określonych warunkach w odniesieniu do ilości wykorzystanych zasobów. Typowe zasoby obejmują: czas, który upłynął, czas procesora, pamięć i pasmo.

4.5.2 Testowanie zachowania w czasie

Testowanie zachowania w czasie mierzy następujące aspekty systemu (lub oprogramowania) w określonych warunkach działania:
- czas, jaki upłynął od otrzymania żądania do pierwszej odpowiedzi (tj. czas do rozpoczęcia odpowiedzi, a nie czas do zakończenia żądanej czynności), zwany również czasem odpowiedzi,
- czas od rozpoczęcia czynności do jej zakończenia, zwany również czasem przetwarzania,
- liczba zakończonych czynności na jednostkę czasu (np. liczba operacji na bazie danych na sekundę), zwana również przepustowością.

Dla wielu systemów maksymalne czasy odpowiedzi dla różnych funkcji systemu są określone przez wymagania. W takich przypadkach czas odpowiedzi to czas do rozpoczęcia odpowiedzi, powiększony o czas przetwarzania. Gdy system musi wykonać pewną liczbę kroków (np. w przetwarzaniu potokowym, ang. *pipeline*), aby zakończyć działanie, przydatne może być zmierzenie czasu potrzebnego na każdy krok i przeanalizowanie wyników w celu określenia, czy jeden lub więcej kroków stanowi wąskie gardło.

4.5.3 Testowanie zużycia zasobów

Testowanie zużycia zasobów mierzy następujące aspekty systemu (lub oprogramowania) w określonych warunkach operacyjnych:

- wykorzystanie procesora, zwykle jako procent dostępnego czasu procesora,
- wykorzystanie pamięci, zwykle jako procent dostępnej pamięci,
- wykorzystanie urządzeń wejścia/wyjścia, zwykle jako procent dostępnego czasu pracy urządzeń wejścia/wyjścia,
- wykorzystanie pasma, zwykle jako procent dostępnej szerokości pasma.

4.5.4 Testowanie pojemności

Testowanie pojemności mierzy maksymalne limity dla następujących aspektów systemu (lub oprogramowania) w określonych warunkach działania:

- transakcje przetwarzane w jednostce czasu (np. maksymalnie 687 słów tłumaczonych na minutę),
- użytkownicy mający jednocześnie dostęp do systemu (np. maksymalnie 1.223 użytkowników),
- nowi użytkownicy dodawani w celu uzyskania dostępu do systemu w jednostce czasu (np. maksymalnie 400 użytkowników dodawanych na sekundę).

4.5.5 Typowe aspekty testowania wydajnościowego

Podczas testowania zachowania w czasie, zużycia zasobów lub pojemności zwykle wykonywanych jest kilka pomiarów, a średnia jest używana jako raportowana miara; jest to spowodowane tym, że mierzony czas może się wahać w zależności od innych zadań w tle, które system może wykonywać. W niektórych sytuacjach pomiary będą traktowane w sposób bardziej drobiazgowy (np. przy użyciu wariancji lub innych miar statystycznych) lub wartości odstające (ang. *outliers*) będą badane i odrzucane, jeśli jest to właściwe.

Analiza dynamiczna (patrz sekcja 3.3.4.) może być wykorzystana do identyfikacji komponentów powodujących wąskie gardło, pomiaru zużycia zasobów w testowaniu zużycia zasobów oraz pomiaru maksymalnych limitów w testowaniu pojemności.

4.5.6 Rodzaje testowania wydajnościowego

Testowanie wydajnościowe różni się od większości innych form testowania tym, że może mieć dwa odrębne cele. Pierwszym z nich jest określenie, czy testowane oprogramowanie spełnia określone kryteria akceptacji. Na przykład określenie, czy system wyświetla żądaną stronę internetową w ciągu maksymalnie 4 sekund, zdefiniowanych w kryteriach akceptacji. Drugim celem jest dostarczenie informacji twórcom systemu, które pomogą im poprawić jego wydajność. Na przykład wykrycie wąskich gardeł i zidentyfikowanie, na które części architektury systemu ma negatywny wpływ niespodziewanie duża liczba użytkowników uzyskujących jednoczesny dostęp do systemu.

Rodzaje testowania wydajnościowego opisane w sekcjach 4.5.2, 4.5.3 i 4.5.4 mogą być wykorzystane do określenia, czy testowane oprogramowanie spełnia określone kryteria akceptacji. Są one również używane do pomiaru wartości bazowych, które są używane do późniejszych porównań, gdy system jest zmieniany. Rodzaje testów wydajnościowych opisane poniżej są częściej używane do dostarczenia informacji programistom, jak system reaguje w różnych warunkach użytkowania.

4.5.6.1. Testowanie obciążenia

Testowanie obciążenia skupia się na zdolności systemu do obsługi różnych obciążeń. Obciążenia te są zazwyczaj definiowane w kategoriach liczby użytkowników mających dostęp do systemu jednocześnie lub liczby współbieżnie działających procesów i mogą być definiowane jako profile operacyjne (więcej szczegółów na temat profili operacyjnych znajduje się w podrozdziale 4.9.). Obsługa tych obciążeń jest zazwyczaj mierzona pod względem zachowania w czasie i zużycia zasobów (np. określenie wpływu podwojenia liczby użytkowników na czas odpowiedzi). Podczas przeprowadzania testów obciążeniowych normalną praktyką jest rozpoczęcie od niskiego obciążenia i stopniowe zwiększanie go przy jednoczesnym pomiarze zachowania w czasie i zużycia zasobów. Typowe informacje z testów obciążeniowych, które mogą

być przydatne dla programistów, to nieoczekiwane zmiany w czasach odpowiedzi lub zużyciu zasobów systemowych, kiedy system obsługiwał określone obciążenie.

4.5.6.2. Testowanie przeciążające

Istnieją dwa rodzaje testów przeciążających; pierwszy z nich jest podobny do testów obciążeniowych, a drugi jest formą testów odpornościowych (ang. *robustness testing*).

W pierwszym przypadku testy obciążeniowe są zazwyczaj wykonywane z obciążeniem początkowo ustawionym na maksymalne oczekiwane, a następnie jest ono zwiększane aż do momentu, gdy system nie ulegnie awarii (np. czasy odpowiedzi stają się nieracjonalnie długie lub system się zawiesza). Czasami, zamiast zmuszać system do awarii, stosuje się wysokie obciążenie w celu przeciążenia systemu, a następnie zmniejsza się obciążenie do normalnego poziomu i sprawdza się, czy system odzyskał wydajność na poziomie sprzed stanu przeciążenia.

W drugim przypadku testy wydajnościowe są przeprowadzane z systemem celowo zagrożonym poprzez ograniczenie dostępu do oczekiwanych zasobów (np. zmniejszenie dostępnej pamięci lub pasma). Wyniki testów przeciążających mogą dostarczyć programistom wiedzę na temat tego, które aspekty systemu są najbardziej krytyczne (tzw. słabe ogniwa) i mogą wymagać aktualizacji.

4.5.6.3. Testowanie skalowalności

Skalowalny system może dostosować się do różnych obciążeń. Na przykład skalowalna strona internetowa może używać więcej serwerów backendowych w miarę wzrostu zapotrzebowania i mniej, gdy zapotrzebowanie maleje. Testowanie skalowalności jest podobne do testowania obciążenia, ale umożliwia testowanie zdolności systemu do skalowania w górę i w dół w obliczu zmieniających się obciążeń (np. więcej użytkowników niż obecny sprzęt może obsłużyć).

Sylabus Certyfikowany Tester – Tester Wydajności [CT_PT_SYL] opisuje dalsze rodzaje testów wydajnościowych.

4.5.7 Planowanie testów wydajnościowych

Ogólnie rzecz biorąc, przy planowaniu testów wydajnościowych szczególne znaczenie mają następujące zagadnienia:

- Czas. Testy wydajnościowe często wymagają zaimplementowania całego systemu i uruchomienia na reprezentatywnym środowisku testowym, co oznacza, że są zazwyczaj wykonywane jako część testów systemowych.
- Przeglądy. Przeglądy kodu, w szczególności te, które skupiają się na interakcji z bazą danych, interakcji komponentów i obsłudze błędów, mogą zidentyfikować problemy związane z wydajnością (w szczególności dotyczące logiki ”czekaj i spróbuj ponownie” oraz nieefektywnych zapytań) i powinny być zaplanowane tak, aby odbyły się tak szybko, jak tylko kod jest dostępny (tj. przed testami dynamicznymi).
- Wczesne testowanie. Niektóre testy wydajnościowe (np. określenie wykorzystania procesora przez krytyczny komponent) mogą być zaplanowane jako część testowania modułowego. Komponenty zidentyfikowane jako wąskie gardło w testach wydajnościowych mogą być aktualizowane i ponownie testowane w izolacji w ramach testowania modułowego.
- Zmiany w architekturze. Niekorzystne wyniki testów wydajnościowych mogą czasami prowadzić do zmian w architekturze systemu. Jeżeli wyniki testów wydajnościowych mogą sugerować tak duże zmiany w systemie, testy wydajnościowe powinny rozpocząć się jak najwcześniej, aby zmaksymalizować czas na rozwiązanie takich problemów.
- Koszty. Narzędzia i środowiska testowe mogą być kosztowne, co oznacza, że mogą być wykorzystywane tymczasowe środowiska testowe w chmurze, a licencje na narzędzia "doładowywane" w razie potrzeby. W takich przypadkach planowanie testów zazwyczaj wymaga optymalizacji czasu spędzonego na przeprowadzaniu testów, aby zminimalizować koszty.
- Środowisko testowe. Środowisko testowe musi być jak najbardziej reprezentatywne dla środowiska produkcyjnego, w przeciwnym razie rośnie trudność skalowania wyników testów wydajnościowych ze środowiska testowego do oczekiwanego środowiska produkcyjnego.

- Kryteria wyjścia. Wymagania dotyczące wydajności mogą być czasami trudne do uzyskania od klienta, dlatego często są określane na podstawie wartości bazowych z poprzednich lub podobnych systemów. W przypadku systemów wbudowanych związanych z bezpieczeństwem, niektóre wymagania, takie jak maksymalna ilość używanego procesora i pamięci, mogą być określone przez normy prawne.
- Narzędzia. Narzędzia do generowania obciążenia są często wymagane do wsparcia testów wydajnościowych. Na przykład, weryfikacja skalowalności popularnego serwisu internetowego może wymagać symulacji setek tysięcy wirtualnych użytkowników. Narzędzia symulujące ograniczenia zasobów są również szczególnie przydatne w testach przeciążających (testach warunków skrajnych). Należy zwrócić uwagę na to, aby wszelkie narzędzia nabyte w celu wsparcia testów były kompatybilne z protokołami komunikacyjnymi używanymi przez testowany system.

Sylabus Certyfikowany Tester – Tester Wydajności [CT_PT_SYL] zawiera dalsze szczegóły dotyczące planowania testów wydajnościowych.

4.5.8 Specyfikacja testów wydajnościowych

Testowanie wydajności opiera się w dużej mierze na testowaniu systemu w określonych warunkach użytkowania. Dla takich testów wymagany jest profil operacyjny, który określa, w jaki sposób system ma być używany. Więcej szczegółów na temat profili operacyjnych znajduje się w podrozdziale 4.9.

W przypadku testów wydajnościowych często konieczna jest zmiana poziomu obciążenia systemu poprzez modyfikację części profilu operacyjnego, w celu symulowania zmiany w stosunku do oczekiwanego użycia systemu. Na przykład podczas testowania pojemności, zazwyczaj będzie konieczne nadpisanie profilu operacyjnego w obszarze zmiennej poddawanej testom pojemności (np. zwiększanie liczby użytkowników korzystających z systemu, do momentu, gdy system przestaje odpowiadać, w celu ustalenia maksymalnej dopuszczalnej pojemności dotyczącej liczby jednoczesnych użytkowników). Podobnie, podczas testowania obciążenia, wolumeny transakcji mogą być stopniowo zwiększane.

Sylabus Certyfikowany Tester – Tester Wydajności [CT_PT_SYL] zawiera dalsze szczegóły dotyczące projektowania testów wydajnościowych.

4.6 Testowanie utrzymywalności

Czas poświęcany na utrzymanie oprogramowania jest często dłuższy niż czas jego wytworzenia. Aby zapewnić jak największą efektywność procesu utrzymania, wykonuje się testy utrzymywalności (ang. *maintainability testing*), które pozwalają zmierzyć stopień łatwości analizowania kodu, wprowadzania w nim zmian oraz testowania, modularyzacji i ponownego wykorzystania. Testowanie utrzymywalności nie powinno być mylone z testowaniem pielęgnacyjnym, które jest wykonywane w celu przetestowania zmian wprowadzonych do działającego oprogramowania.

Wśród celów związanych z utrzymywalnością, jakie chcą uzyskać interesariusze (np. właściciele lub operatorzy oprogramowania), znajdują się:
- minimalizacja kosztów posiadania lub eksploatacji oprogramowania,
- ograniczenie przestojów niezbędnych do pielęgnacji oprogramowania.

Testy utrzymywalności powinny zostać uwzględnione w podejściu do testowania, jeśli spełniony jest co najmniej jeden z następujących warunków:
- Prawdopodobne jest wprowadzanie zmian w oprogramowaniu w wersji produkcyjnej (np. w celu usunięcia defektów lub wprowadzenia zaplanowanych aktualizacji).
- Interesariusze uznają, że korzyści wynikające z osiągnięcia celów w obszarze utrzymywalności w cyklu życia oprogramowania przeważają nad kosztami wykonania testów utrzymywalności i wprowadzenia niezbędnych zmian.
- Czynniki ryzyka związane z niską utrzymywalnością oprogramowania (np. długie czasy reakcji na defekty zgłaszane przez użytkowników i/lub klientów) uzasadniają przeprowadzenie takich testów.

4.6.1 Statyczne i dynamiczne testowanie utrzymywalności

Technikami stosowanymi w trakcie statycznego testowania utrzymywalności są m.in. analiza statyczna i przeglądy, zgodnie z opisem przedstawionym w podrozdziałach 3.2. i 5.2. Testowanie utrzymywalności należy rozpocząć od razu po udostępnieniu dokumentacji projektowej i kontynuować je w trakcie prac związanych z implementacją kodu. Utrzymywalność to cecha związana z kodem i dokumentacją poszczególnych modułów, dlatego można ją ocenić na wczesnym etapie cyklu życia oprogramowania, bez konieczności oczekiwania na udostępnienie kompletnego, działającego systemu.

Dynamiczne testowanie utrzymywalności skupia się na udokumentowanych procedurach utrzymania poszczególnych aplikacji (np. opisujących wykonywanie aktualizacji oprogramowania). Scenariusze pielęgnacyjne są wykorzystywane jako przypadki testowe w celu sprawdzenia, czy udokumentowane procedury pozwalają uzyskać wymagane poziomy usług. Taka forma testowania ma szczególne znaczenie w sytuacji, gdy używana infrastruktura jest złożona, a procedury wsparcia obejmują wiele działów lub organizacji. Testowanie tego rodzaju można wykonać w ramach produkcyjnych testów akceptacyjnych.

4.6.2 Podcharakterystyki jakościowe utrzymywalności

Utrzymywalność systemu może być wyrażona jako:

- analizowalność,
- modyfikowalność,
- testowalność.

Do czynników, które mają wpływ na powyższe charakterystyki, należą dobre praktyki związane z programowaniem (np. tworzenie komentarzy, odpowiednie nazewnictwo zmiennych i umieszczanie wcięć) oraz dostępność dokumentacji technicznej (np. specyfikacji projektowej systemu i specyfikacji interfejsów).

Inne ważne podcharakterystyki jakościowe utrzymywalności [ISO 25010] to:

- modułowość,
- możliwość ponownego użycia.

Modułowość może być testowana za pomocą analizy statycznej (patrz sekcja 3.2.3.). Testowanie możliwości ponownego użycia może przybrać formę przeglądów architektury (patrz Rozdział 5.).

4.7 Testowanie przenaszalności

4.7.1 Wstęp

Ogólnie rzecz biorąc, testowanie przenaszalności odnosi się do łatwości przenoszenia systemu lub modułu oprogramowania do docelowego środowiska (zarówno po raz pierwszy, jak i z istniejącego środowiska), dostosowania do nowego środowiska lub zastąpienia innej jednostki.

Standard ISO 25010 [ISO 25010] opisuje następujące podcharakterystyki przenaszalności:

- Instalowalność,
- zdolność adaptacyjna,
- zastępowalność.

Testowanie przenaszalności można zacząć od poszczególnych modułów (m.in. weryfikując zastępowalność danego modułu, np. przejście z jednego systemu zarządzania bazą danych na inny) i zwiększać jego zakres w miarę udostępniania kolejnych fragmentów kodu. Instalowalność czasami można przetestować dopiero wówczas, gdy działają wszystkie moduły produktu.

Przenaszalność powinna zostać uwzględniona w projekcie i wbudowana w produkt, dlatego należy ją wziąć pod uwagę na wczesnych etapach faz projektowania i tworzenia architektury. Przeglądy projektu

i architektury stanowią szczególnie skuteczną metodę identyfikowania wymagań dotyczących przenaszalności i potencjalnych problemów (np. zależności od konkretnego systemu operacyjnego).

4.7.2 Testowanie instalowalności

Testowanie instalowalności przeprowadza się na oprogramowaniu oraz na procedurach używanych do jego zainstalowania w docelowym środowisku. Może to na przykład obejmować oprogramowanie przeznaczone do zainstalowania systemu produkcyjnego albo kreatora instalacji używanego do zainstalowania produktu na komputerze klienta.

Typowymi celami testowania instalowalności są:

- Sprawdzenie, czy oprogramowanie może zostać zainstalowane po zastosowaniu instrukcji podanych w podręczniku instalacji (łącznie z uruchomieniem ewentualnych skryptów instalacyjnych) lub za pomocą kreatora instalacji. Obejmuje to również sprawdzenie opcji instalacji dla różnych konfiguracji programowo/sprzętowych i różnych stopni instalacji (np. instalacji początkowej i aktualizacji).
- Sprawdzenie, czy awarie występujące podczas instalacji (np. niemożność załadowania konkretnej biblioteki DLL) są obsługiwane poprawnie przez oprogramowanie instalacyjne i czy system nie jest pozostawiany w stanie niezdefiniowanym (np. z oprogramowaniem częściowo zainstalowanym albo w niepoprawnej konfiguracji).
- Przetestowanie możliwości wykonania częściowej instalacji lub deinstalacji.
- Przetestowanie, czy kreator instalacji jest w stanie zidentyfikować niepoprawne platformy sprzętowe i konfiguracje systemu operacyjnego.
- Sprawdzenie, czy proces instalacji może zostać zakończony w podanym czasie (liczba minut) lub w ramach założonej liczby kroków.
- Sprawdzenie, czy można przejść na starszą wersję oprogramowania lub deinstalować oprogramowanie.

Testowanie przydatności funkcjonalnej wykonuje się zwykle po instalacji w celu wykrycia ewentualnych defektów powstałych podczas instalacji (np. niepoprawnych konfiguracji i braku dostępności funkcji). Standardowo równolegle z testowaniem instalowalności prowadzone jest testowanie użyteczności, na przykład w celu sprawdzenia, czy użytkownicy otrzymują podczas instalacji zrozumiałe instrukcje, odpowiedzi i komunikaty o błędach.

4.7.3 Testowanie zdolności adaptacyjnej

Testowanie zdolności adaptacyjnej umożliwia sprawdzenie, czy dana aplikacja może funkcjonować poprawnie we wszystkich założonych środowiskach docelowych (sprzęt, oprogramowanie, warstwa pośrednia, system operacyjny itd.). W ramach specyfikacji testów zdolności adaptacyjnej należy zidentyfikować, skonfigurować i udostępnić zespołowi testowemu odpowiednie środowiska docelowe. Następnie takie środowiska zostają przetestowane za pomocą wybranych funkcjonalnych przypadków testowych sprawdzających różnego rodzaju moduły obecne w środowisku.

Zdolność adaptacyjna może odnosić się do możliwości przeniesienia oprogramowania do różnych określonych środowisk przez wykonanie zdefiniowanej procedury. Testy mogą służyć do weryfikacji tej procedury.

4.7.4 Testowanie zastępowalności

Testowanie zastępowalności koncentruje się na zdolności modułu oprogramowania do zastąpienia istniejącego modułu oprogramowania w systemie. Może być ono szczególnie istotne w przypadku systemów, w których do obsługi pewnych modułów stosowane jest oprogramowanie do powszechnej sprzedaży (ang. *commercial off-the-shelf* – COTS) lub w przypadku aplikacji Internetu rzeczy (IoT).

Testy zastępowalności mogą być wykonywane równolegle z testami integracji funkcjonalności systemu, jeśli do integracji w pełnym systemie dostępny jest więcej niż jeden opcjonalny moduł. Zastępowalność może być również oceniana w ramach przeglądu technicznego lub inspekcji na poziomie architektury

i projektu, gdzie szczególną wagę przywiązuje się do jasnego zdefiniowania interfejsów komponentu, który może być użyty jako zamiennik.

4.8 Testowanie kompatybilności

4.8.1 Wstęp

W testowaniu kompatybilności brane są pod uwagę następujące aspekty [ISO 25010]:

- współistnienie,
- współdziałanie.

4.8.2 Testowanie współistnienia

Systemy, które nie są ze sobą powiązane, nazywane są współistniejącymi, jeśli mogą zostać uruchomione w tym samym środowisku (np. na tym samym sprzęcie) bez wzajemnego wpływu na ich działanie (np. bez konfliktów dotyczących zasobów). Testowanie współistnienia należy wykonać wtedy, gdy nowe lub zaktualizowane oprogramowanie ma zostać wdrożone w środowiskach, które zawierają już pewne zainstalowane aplikacje.

Problemy w tym obszarze mogą wystąpić wówczas, gdy aplikacja jest testowana w środowisku, w którym jest jedyną zainstalowaną aplikacją (i w związku z tym braku kompatybilności nie da się wykryć), a następnie zostanie przeniesiona do innego środowiska (np. produkcyjnego), w którym funkcjonują także inne aplikacje.

Typowe cele testowania współistnienia:

- Ocena możliwego niekorzystnego wpływu na funkcjonalność w przypadku ładowania aplikacji w tym samym środowisku (np. konflikt użycia zasobów, gdy na serwerze jest uruchomionych wiele aplikacji).
- Ocena wpływu instalacji poprawek i aktualizacji systemu operacyjnego na działanie aplikacji.

Kwestie związane ze współistnieniem należy uwzględnić podczas planowania docelowego środowiska produkcyjnego, jednak same testy zwykle wykonuje się dopiero po zakończeniu testowania systemowego.

4.9 Profile operacyjne

Profile operacyjne są używane jako część specyfikacji testów dla wielu rodzajów testów niefunkcjonalnych, w tym testów niezawodności i wydajności. Są one szczególnie użyteczne, gdy testowane wymaganie zawiera ograniczenie "w określonych warunkach", ponieważ mogą być użyte do zdefiniowania tych warunków.

Profil operacyjny definiuje wzorzec użycia systemu, zazwyczaj w kategoriach użytkowników systemu i operacji wykonywanych przez system. Użytkownicy są zazwyczaj określani pod względem tego, ilu z nich ma korzystać z systemu (i w jakich godzinach), a także ewentualnie ich rodzaju (np. użytkownik główny, użytkownik drugorzędny). Różne operacje, które mają być wykonywane przez system, są zazwyczaj określone wraz z ich częstotliwością (i prawdopodobieństwem wystąpienia). Informacje te mogą być uzyskane za pomocą narzędzi monitorujących (gdy rzeczywista lub podobna aplikacja jest już dostępna) lub poprzez przewidywanie użycia na podstawie algorytmów lub oszacowań dostarczonych przez organizację biznesową.

Narzędzia mogą być używane do generowania danych wejściowych do testów w oparciu o profil operacyjny, często wykorzystując podejście, które generuje dane wejściowe do testów w sposób pseudo-losowy. Takie narzędzia mogą być używane do tworzenia "wirtualnych" lub symulowanych użytkowników w ilościach, które odpowiadają profilowi operacyjnemu (np. do testowania niezawodności i dostępności) lub go przekraczają (np. do testowania przeciążającego lub wydajnościowego). Więcej szczegółów na temat tych narzędzi znajduje się w sekcji 6.2.3.

5. Przeglądy — 165 minut

Słowa kluczowe
przegląd, przegląd techniczny

Cele nauczania dotyczące przeglądów

5.1 Zadania Technicznego Analityka Testów w trakcie przeglądów

TTA 5.1.1. (K2) Kandydat potrafi wyjaśnić, dlaczego przygotowanie do przeglądu jest istotne dla Technicznego Analityka Testów

5.2 Korzystanie z list kontrolnych podczas przeglądów

TTA 5.2.1. (K4) Kandydat potrafi przeanalizować projekt architektury i zidentyfikować problemy zgodnie z listą kontrolną podaną w sylabusie

TTA 5.2.2. (K4) Kandydat potrafi przeanalizować fragment kodu lub pseudokodu i zidentyfikować problemy zgodnie z listą kontrolną podaną w sylabusie

5.1 Zadania Technicznego Analityka Testów w trakcie przeglądów

Techniczni analitycy testów muszą aktywnie uczestniczyć w procesie przeglądu technicznego, przedstawiając specyficzny (wyjątkowy) punkt widzenia na poruszane zagadnienia. Wszyscy uczestnicy przeglądu powinni zostać przeszkoleni w zakresie przeglądów formalnych tak, aby rozumieli swoją odpowiednią rolę, a ponadto wszystkim musi zależeć na osiągnięciu korzyści wynikających z dobrze przeprowadzonego przeglądu technicznego. Obejmuje to zachowanie konstruktywnych kontaktów roboczych z autorami podczas zgłaszania komentarzy i ich omawiania. Szczegółowy opis przeglądów technicznych, łącznie z przykładami wielu list kontrolnych, można znaleźć w pracy [Wiegers02]. Techniczni analitycy testów zwykle biorą udział w przeglądach technicznych i inspekcjach, w trakcie których są w stanie przedstawić perspektywę operacyjną (dotyczącą zachowania systemu), która mogła zostać pominięta przez programistów. Ponadto techniczni analitycy testów odgrywają ważną rolę w definiowaniu, realizowaniu i utrzymywaniu list kontrolnych przeglądów oraz dostarczaniu informacji o ważności defektów.

Niezależnie od typu dokonywanego przeglądu Techniczny Analityk Testów musi zaplanować odpowiednio dużo czasu na przygotowanie. Przygotowanie obejmuje czas potrzebny na przejrzenie produktu, na sprawdzenie powiązanej dokumentacji pod kątem spójności oraz na ustalenie, czego może brakować w danym produkcie pracy. Bez odpowiedniego przygotowania przegląd może zostać sprowadzony wyłącznie do prostych prac redakcyjnych, a w rezultacie nie być prawdziwym przeglądem. Dobrze przeprowadzony przegląd obejmuje zrozumienie tego, co zostało napisane, ustalenie, czego brakuje, sprawdzenie poprawności pod względem technicznym oraz tego, czy dany produkt jest spójny z innymi produktami (już opracowanymi albo dopiero przygotowywanymi). Na przykład podczas przeglądu planu testów na poziomie testów integracyjnych Techniczny Analityk Testów musi również uwzględnić integrowane elementy. Czy są 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 danego produktu pracy podlegającego przeglądowi, ale należy w nim również uwzględnić interakcje tego produktu z innymi elementami systemu.

5.2 Korzystanie z list kontrolnych podczas przeglądów

Podczas przeglądów używa się list kontrolnych przypominających uczestnikom o konieczności zweryfikowania konkretnych elementów. Pomagają one również w wyeliminowaniu czynnika osobistego z przeglądu: „używamy tej samej listy we wszystkich przeglądach, nie tylko w odniesieniu do twojego produktu pracy”. Listy kontrolne mogą być ogólne, do zastosowania we wszelkiego rodzaju przeglądach, lub skoncentrowane na określonych charakterystykach jakościowych albo obszarach. Na przykład ogólna lista kontrolna może zawierać takie punkty, jak sprawdzenie odpowiedniego użycia pojęć „będzie” i „powinien”, weryfikacja właściwego formatowania oraz inne podobne elementy dotyczące zgodności z szablonem. Listy kontrolne mogą koncentrować się na zagadnieniach związanych z zabezpieczeniami lub wydajnością.

Najbardziej przydatne są listy kontrolne sukcesywnie tworzone przez konkretne jednostki organizacyjne, ponieważ uwzględniają:

- charakter produktu,
- lokalne środowisko wytwórcze:
  - personel,
  - narzędzia,
  - priorytety,
- historię poprzednich sukcesów i defektów,
- specyficzne problemy (np. dotyczące wydajności i zabezpieczeń).

Listy powinny zostać dostosowane do potrzeb organizacji, a być może także do potrzeb konkretnego projektu. Listy kontrolne w niniejszym rozdziale należy traktować jako przykłady.

5.2.1 Przeglądy architektury

Architektura oprogramowania określa fundamentalne koncepcje lub właściwości systemu, opisujące jego elementy, wzajemne relacje oraz zasady projektowania i rozwoju systemu [ISO 42010].

Listy kontrolne[1] stosowane w trakcie przeglądów architektury dla zachowania w czasie stron webowych mogą na przykład zawierać weryfikację poprawnej implementacji następujących elementów (lista pochodzi ze strony [Web-2]):

- zestawianie połączeń — skrócenie narzutu czasowego związanego z nawiązywaniem połączeń z bazą danych poprzez utworzenie współużytkowanej puli połączeń,
- równoważenie obciążenia — równomierne rozłożenie obciążenia między elementy zestawu zasobów,
- przetwarzanie rozproszone,
- buforowanie — używanie lokalnej kopii danych w celu skrócenia czasu dostępu,
- tworzenie obiektów z opóźnieniem (tzw. leniwe inicjowanie, ang. *lazy instantiation*),
- współbieżność transakcji,
- odseparowanie procesów przetwarzania transakcyjnego na bieżąco (ang. *Online Transactional Processing* - OLTP) i analitycznego przetwarzania na bieżąco (ang. *Online Analytical Processing* - OLAP),
- replikacja danych.

5.2.2 Przeglądy kodu

Listy kontrolne przeglądów kodu są z założenia niskopoziomowe i sprawdzają się najlepiej, jeśli dotyczą konkretnego języka programowania. Uwzględnienie antywzorców dotyczących kodu może okazać się pomocne, szczególnie w przypadku mniej doświadczonych programistów.

Na listach kontrolnych[2] używanych w trakcie przeglądów kodu można uwzględnić poniższe zagadnienia:

1. Struktura
   - Czy kod w pełny i prawidłowy sposób implementuje projekt?
   - Czy kod jest zgodny z odpowiednimi standardami kodowania?
   - Czy kod ma poprawną strukturę, jednolity styl i jednolite formatowanie?
   - Czy istnieją niewywoływane lub niepotrzebne procedury i/lub nieosiągalne fragmenty kodu?
   - Czy w kodzie pozostały zaślepki i procedury testowe?
   - Czy jakieś fragmenty kodu mogą zostać zastąpione wywołaniami do zewnętrznych modułów wielokrotnego użytku lub funkcji z biblioteki?
   - Czy istnieją bloki powtarzającego się kodu, które można zastąpić pojedynczą procedurą?
   - Czy wykorzystanie pamięci jest efektywne?
   - Czy jako stałe używane są wartości symboliczne, a nie „magiczne liczby” lub stałe łańcuchowe (ang. *string constans*)?
   - Czy istnieją moduły o zbyt dużym stopniu złożoności, które należy zrestrukturyzować lub podzielić na wiele modułów?

2. Dokumentacja
   - Czy kod jest udokumentowany w jasny i prawidłowy sposób, a styl komentarzy umożliwia proste wprowadzanie zmian?
   - Czy wszystkie komentarze mają treść zgodną z kodem?
   - Czy dokumentacja jest zgodna z obowiązującymi standardami?

3. Zmienne
   - Czy wszystkie zmienne są właściwie zdefiniowane i mają znaczące, spójne i zrozumiałe nazwy?
   - Czy istnieją nadmiarowe lub nieużywane zmienne?

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

4. Operacje arytmetyczne
   - Czy w kodzie unika się porównywania liczb zmiennoprzecinkowych?
   - Czy w kodzie w systematyczny sposób unika się błędów zaokrągleń?
   - Czy w kodzie nie pojawiają się operacje dodawania i odejmowania liczb o różniących się znacznie rzędach wielkości?
   - Czy sprawdzane są dzielniki (czy są różne od zera i nie mają zaburzonych wartości)?

5. Pętle i gałęzie
   - Czy wszystkie pętle, gałęzie kodu i konstrukcje logiczne są pełne, poprawne i prawidłowo zagnieżdżone?
   - Czy w łańcuchach instrukcji IF-ELSEIF najczęstsze przypadki są sprawdzane na początku?
   - Czy w bloku instrukcji IF-ELSEIF lub CASE sprawdzane są wszystkie przypadki i czy są uwzględnione klauzule ELSE lub DEFAULT?
   - Czy każda instrukcja CASE ma klauzulę DEFAULT?
   - Czy warunki zakończenia pętli są oczywiste i zawsze osiągalne?
   - Czy indeksy (w tym indeksy tablic) są poprawnie zainicjowane tuż przed wejściem do pętli?
   - Czy któreś z instrukcji znajdujących się w pętlach można umieścić poza pętlami?
   - Czy w pętli unika się manipulowaniem zmienną sterującą ani czy nie korzysta się z niej w momencie wyjścia z pętli?

6. Programowanie defensywne
   - Czy sprawdzane jest przekraczanie wartości granicznych w tablicy, rekordzie lub pliku przez indeksy, wskaźniki i indeksy tabel?
   - Czy sprawdzana jest poprawność i kompletność zaimportowanych danych i argumentów wejściowych?
   - Czy wszystkie zmienne wyjściowe mają przypisane wartości?
   - Czy w każdej instrukcji używany jest właściwy element danych?
   - Czy każdy przydział pamięci jest zwalniany?
   - Czy w przypadku dostępu do urządzeń zewnętrznych stosowane są limity czasu lub mechanizmy obsługi błędów?
   - Czy przed próbą dostępu do pliku kod sprawdza, czy plik istnieje?
   - Czy w momencie zakończenia programu wszystkie pliki i urządzenia pozostają w prawidłowym stanie?

6. Narzędzia testowe i automatyzacja testów — 180 minut

Słowa kluczowe
emulator, posiew usterek, rejestruj i odtwórz, symulator, testowanie oparte na słowach kluczowych, testowanie oparte na modelu (MBT), testowanie sterowane danymi, wstrzykiwanie błędów, wykonywanie testu

Cele nauczania dotyczące narzędzi testowych i automatyzacji testów

6.1 Definiowanie projektu automatyzacji testów

TTA-6.1.1. (K2) Kandydat potrafi omówić czynności wykonywane przez Technicznego Analityka Testów podczas tworzenia projektu automatyzacji testów

TTA-6.1.2. (K2) Kandydat potrafi omówić różnice między automatyzacją sterowaną danymi a automatyzacją opartą na słowach kluczowych

TTA-6.1.3. (K2) Kandydat potrafi omówić często występujące problemy techniczne, z powodu których w projektach automatyzacji nie udaje się uzyskać zaplanowanego zwrotu z inwestycji

TTA-6.1.4. (K3) Kandydat potrafi utworzyć słowa kluczowe na podstawie danego procesu biznesowego

6.2 Kategorie narzędzi testowych

TTA-6.2.1. (K2) Kandydat potrafi omówić zastosowanie narzędzi do posiewu usterek i wstrzykiwania błędów

TTA-6.2.2. (K2) Kandydat potrafi omówić główne cechy narzędzi do testów wydajnościowych oraz zagadnienia dotyczące ich wdrażania

TTA-6.2.3. (K2) Kandydat potrafi wyjaśnić ogólne zastosowania narzędzi do testowania stron internetowych

TTA-6.2.4. (K2) Kandydat potrafi wyjaśnić, w jaki sposób narzędzia wspierają praktykę testowania opartego na modelu

TTA-6.2.5. (K2) Kandydat potrafi omówić zastosowanie narzędzi wykorzystywanych do wspierania testów modułowych i procesu budowania

TTA-6.2.6. (K2) Kandydat potrafi omówić zastosowanie narzędzi wykorzystywanych do wspierania testów aplikacji mobilnych

6.1 Definiowanie projektu automatyzacji testów

Narzędzia testowe, a zwłaszcza narzędzia wspierające wykonywanie testów, muszą mieć odpowiednią architekturę i zostać bardzo dobrze zaprojektowane, aby ich zastosowanie było opłacalne. Efektem wdrożenia strategii automatyzacji wykonywania testów bez opracowania właściwej architektury będzie zestaw narzędzi kosztowny w utrzymaniu i niewystarczający do realizacji zamierzonych celów, a uzyskanie zakładanego zwrotu z inwestycji okaże się niemożliwe.

Projekt automatyzacji testów należy traktować jak projekt rozwoju oprogramowania. Oznacza to konieczność opracowania dokumentacji architektury i szczegółowej dokumentacji projektowej, wykonania przeglądów projektu i kodu, przetestowania modułów i przetestowania integracji modułów, a także przeprowadzenia końcowych testów systemowych. Zastosowanie niestabilnego lub niepoprawnego kodu automatyzacji testów może niepotrzebnie wydłużyć lub skomplikować proces testowania.

Techniczny Analityk Testów może wykonywać wiele zadań związanych z automatyzacją wykonywania testów. To między innymi:

- Ustalenie osób odpowiedzialnych za wykonanie testów (we współpracy z Kierownikiem Testów, jeśli to możliwe).
- Wybór odpowiedniego dla organizacji narzędzia, określenie harmonogramu, umiejętności zespołu i wymagań dotyczących pielęgnacji narzędzia (może to w szczególności oznaczać decyzję o opracowaniu własnego narzędzia zamiast dokonywania zakupu).
- Zdefiniowanie wymagań dotyczących interfejsów między narzędziem do automatyzacji a innymi narzędziami, np. narzędziami do zarządzania testami, narzędziami do zarządzania defektami oraz narzędziami do obsługi mechanizmów ciągłej integracji.
- Opracowanie adapterów, które mogą być wymagane do stworzenia interfejsu między narzędziem do wykonywania testów a testowanym oprogramowaniem.
- Wybór podejścia do automatyzacji, tj. opartego na słowach kluczowych lub sterowanego danymi (patrz sekcja 6.1.1.).
- Określenie kosztów wdrożenia (w tym szkoleń) wspólnie z Kierownikiem Testów. W zwinnym wytwarzaniu oprogramowania ten aspekt jest zwykle dyskutowany i uzgadniany w trakcie spotkań poświęconych planowaniu projektu/sprintu z udziałem całego zespołu.
- Określenie harmonogramu projektu automatyzacji i zaplanowanie czasu na utrzymanie systemu.
- Przeszkolenie analityków testów i analityków biznesowych w zakresie korzystania i sposobów dostarczania danych do automatyzacji.
- Określenie sposobu i momentu wykonywania testów automatycznych.
- Określenie sposobów łączenia rezultatów testów automatycznych z rezultatami testów manualnych.

W projektach, w których automatyzacja testowania odgrywa dużą rolę, za realizację wielu z tych zadań może być odpowiedzialny Inżynier Testów Automatycznych (szczegółowe informacje na ten temat można znaleźć w sylabusie Certyfikowany Tester – Inżynier Automatyzacji Testów [CT_TAE_SYL]. Niektóre z zadań natury organizacyjnej może wykonywać Kierownik Testów, zgodnie z potrzebami i preferencjami w danym projekcie. W zwinnym wytwarzaniu oprogramowania powiązanie wymienionych zadań z rolami ma zwykle bardziej elastyczny, mniej formalny charakter.

Działania te i wynikające z nich decyzje mają wpływ na skalowalność i utrzymywalność rozwiązania dla testów automatycznych. Należy poświęcić wystarczającą ilość czasu na przeanalizowanie dostępnych opcji, narzędzi i technologii oraz zrozumienie planów organizacji na przyszłość.

6.1.1 Wybór podejścia do automatyzacji

W niniejszej sekcji omówiono następujące aspekty związane z podejściem do automatyzacji testów:

- automatyzacja za pomocą graficznego interfejsu użytkownika (GUI), interfejsu programowania aplikacji (API) oraz interfejsu wiersza poleceń (CLI),
- testowanie sterowane danymi,
- testowanie oparte na słowach kluczowych,

- obsługa awarii oprogramowania.
- uwzględnianie stanu systemu.

Sylabus Certyfikowany Tester – Inżynier Automatyzacji Testów [CT_TAE_SYL] zawiera dodatkowe informacje na temat wyboru podejścia do automatyzacji.

6.1.1.1. Automatyzacja za pomocą GUI, API oraz CLI

Automatyzacja testów nie ogranicza się do testowania za pośrednictwem graficznego interfejsu użytkownika (GUI). Istnieją również narzędzia do automatyzacji testów na poziomie interfejsu programowania aplikacji (API), za pośrednictwem interfejsu wiersza poleceń (CLI) i innych punktów interfejsu w testowanym oprogramowaniu. Jedną z pierwszych decyzji, jakie musi podjąć Techniczny Analityk Testów, jest wybór interfejsu, z którego najlepiej będzie skorzystać w trakcie automatyzacji testów. Standardowe narzędzia do wykonywania testów wymagają opracowania adapterów do takich interfejsów. Pracochłonność związaną z tworzeniem adapterów należy uwzględnić w trakcie planowania działań.

Jednym z problemów związanych z testowaniem za pośrednictwem graficznego interfejsu użytkownika jest tendencja do wprowadzania w nim zmian w miarę rozwoju oprogramowania. Może to znacznie zwiększyć nakład pracy związany z utrzymaniem kodu do automatyzacji testów, w zależności od sposobu zaprojektowania tego kodu. Na przykład przypadki testowe testów automatycznych (często nazywane skryptami testowymi) utworzone za pomocą funkcji rejestruj i odtwórz mogą po zmianie interfejsu nie działać zgodnie z wymaganiami. Dzieje się tak dlatego, że w nagranym skrypcie zapisane są interakcje z obiektami graficznymi wykonane przez testera po ręcznym wykonaniu oprogramowania, a jeśli poszczególne obiekty są modyfikowane, może zajść potrzeba aktualizacji zarejestrowanego skryptu, by odzwierciedlić te zmiany.

Narzędzia rejestrująco-odtwarzające zapewniają wygodny punkt wyjścia do projektowania skryptów automatyzacji. Tester rejestruje sesję testową i powstały skrypt jest następnie modyfikowany w celu zwiększenia utrzymywalności kodu (np. poprzez zastąpienie fragmentów skryptu wywołaniami funkcji wielokrotnego użytku).

6.1.1.2. Zastosowanie podejścia „testowanie sterowane danymi”

Dane używane w poszczególnych testach mogą być zależne od testowanego oprogramowania, chociaż wykonywane kroki testu są wirtualnie identyczne (np. podczas testowania obsługi błędów w polu wejściowym za pomocą wprowadzania wielu niepoprawnych wartości i sprawdzaniu zwracanego dla każdej z nich błędu). Opracowywanie i utrzymywanie odrębnych skryptów testów automatycznych dla każdej testowanej wartości jest nieefektywne. Często stosowanym rozwiązaniem technicznym tego problemu jest przeniesienie danych ze skryptów do zewnętrznej składnicy, na przykład arkusza kalkulacyjnego lub bazy danych. Tworzy się funkcje uzyskujące dostęp do konkretnych danych dla każdego wykonania skryptu testowego, dzięki czemu jeden skrypt może obsłużyć zestaw danych testowych zawierający wartości wejściowe i oczekiwane wartości wynikowe (np. wartości wyświetlane w polu tekstowym lub komunikaty o błędach). Takie podejście nazywamy testowaniem sterowanym danymi.

Przy zastosowaniu tego podejścia, oprócz skryptów testowych, które przetwarzają dostarczone dane, tworzy się jarzmo testowe i infrastrukturę niezbędną do wykonania skryptu lub zestawu skryptów. Rzeczywiste dane przechowywane w arkuszu lub bazie danych są tworzone przez analityków testów, którzy znają funkcje biznesowe realizowane przez oprogramowanie. W zwinnym wytwarzaniu oprogramowania, w definiowaniu danych, zwłaszcza na potrzeby testów akceptacyjnych, może uczestniczyć także przedstawiciel strony biznesowej (np. właściciel produktu). Taki podział pracy pozwala osobom odpowiedzialnym za opracowanie skryptów testowych (np. technicznym analitykom testów) skoncentrować się na implementacji skryptów automatyzacji, podczas gdy właścicielem samego testu pozostaje Analityk Testów. W większości przypadków za uruchomienie skryptów testowych po ich zaimplementowaniu i przetestowaniu odpowiada właśnie Analityk Testów.

6.1.1.3. Zastosowanie podejścia „testowanie oparte na słowach kluczowych”

W innym podejściu, nazywanym testowaniem opartym na słowach kluczowych lub na słowach akcji, dodatkowo oddziela się działania, które mają zostać wykonane na dostarczonych danych testowych, od samego skryptu testowego [Buwalda01]. Aby uzyskać ten stopień rozdzielenia, tworzy się język wysokiego poziomu, który ma raczej charakter opisowy, a nie służy do bezpośredniego uruchamiania kodu. Słowa

kluczowe mogą być zdefiniowane zarówno dla działań wysokiego, jak i niskiego poziomu. Na przykład wśród słów kluczowych związanych z procesem biznesowym mogą się znaleźć słowa „Zaloguj", „UtwórzUżytkownika" i „UsuńUżytkownika". Takie słowa kluczowe opisują działania wysokiego poziomu wykonywane w domenie aplikacji. Działania niższego poziomu oznaczają interakcje z samym interfejsem oprogramowania. Słowa kluczowe, takie jak: „KliknijPrzycisk", „WybierzZListy" albo „PrzejdźDrzewo" służą do testowania możliwości graficznego interfejsu użytkownika, które nie odpowiadają bezpośrednio słowom kluczowym używanym w procesie biznesowym. Słowa kluczowe mogą zawierać parametry, na przykład słowo kluczowe "LogIn" może mieć dwa parametry: nazwa_użytkownika i hasło.

Po zdefiniowaniu słów kluczowych i używanych danych osoba odpowiedzialna za automatyzację testów (np. Techniczny Analityk Testów lub Inżynier Testów Automatycznych) tłumaczy słowa kluczowe związane z procesem biznesowym i działania niższego poziomu na kod automatyzacji testów. Słowa kluczowe i działania, a także używane w testach dane, można zapisać w arkuszach lub wprowadzić za pomocą konkretnych narzędzi obsługujących automatyzację testów opartą na słowach kluczowych. Struktura do testów automatycznych (ang. *test automation framework*) implementuje słowa kluczowe w postaci zestawu wykonywalnych funkcji lub skryptów. Narzędzia odczytują przypadki testowe zapisane za pomocą słów kluczowych i wywołują odpowiednie funkcje testowe lub skrypty, które je implementują. Skrypty wykonywalne są implementowane w sposób wysoce modułowy, aby łatwo je było odwzorowywać na konkretne słowa kluczowe. Do zaimplementowania takich modułowych skryptów konieczna jest umiejętność programowania.

Oddzielenie znajomości logiki biznesowej od rzeczywistego programowania wymaganego do implementacji skryptów automatyzacji testów pozwala w optymalny sposób wykorzystać zasoby potrzebne do testów. Techniczny Analityk Testów będąc osobą odpowiedzialną za automatyzację może skutecznie wykorzystać swoje kompetencje w zakresie programowania, bez konieczności stawania się ekspertem merytorycznym w wielu obszarach biznesowych.

Oddzielenie kodu od danych podlegających modyfikacjom pozwala odizolować proces automatyzacji od wprowadzanych zmian, poprawić ogólną utrzymywalność kodu i zwiększyć wartość zwrotu z inwestycji w automatyzację.

6.1.1.4. Obsługa awarii oprogramowania

W ramach projektu automatyzacji testów należy przewidzieć awarie oprogramowania i określić sposób ich obsługi. Osoba odpowiedzialna za automatyzację musi określić, jak ma działać oprogramowanie wykonujące testy, jeśli wystąpi awaria. Czy awarię należy zarejestrować i kontynuować testy? Czy testy należy przerwać? Czy można obsłużyć wystąpienie awarii za pomocą konkretnego działania (np. kliknięcia przycisku w oknie dialogowym) albo poprzez dodanie opóźnienia do testu? Nieobsłużone awarie oprogramowania mogą nie tylko spowodować problem w wykonywanym w danym momencie teście, ale wpłynąć także na rezultaty kolejnych testów.

6.1.1.5. Uwzględnianie stanu systemu

Istotne jest także uwzględnienie stanu, w jakim system znajduje się na początku i na końcu każdego testu. Może okazać się konieczne przywrócenie systemu do zdefiniowanego stanu po zakończeniu wykonywania testów. Zestaw testów automatycznych da się dzięki temu wielokrotnie uruchamiać bez ręcznej interwencji użytkownika, który musi przywracać system do znanego stanu. Aby to osiągnąć, testy automatyczne powinny na przykład usuwać utworzone dane lub zmieniać status rekordów w bazie. Struktura do testów automatycznych powinna zagwarantować, że zakończenie testów odbędzie się w poprawny sposób (np. nastąpi wylogowanie).

6.1.2 Modelowanie procesów biznesowych na potrzeby automatyzacji

Aby wdrożyć podejście do automatyzacji testów oparte na słowach kluczowych, należy przeprowadzić modelowanie procesów biznesowych, które mają zostać przetestowane, w języku słów kluczowych wysokiego poziomu. Język powinien być intuicyjny dla użytkowników, którymi prawdopodobnie będą analitycy testów biorący udział w projekcie, a w przypadku zwinnego wytwarzania oprogramowania – przedstawiciel strony biznesowej (np. właściciel produktu).

Słowa kluczowe są powszechne używane do opisu wysokopoziomowych interakcji biznesowych z systemem. Na przykład słowo kluczowe „Anuluj_Zamówienie” może oznaczać konieczność sprawdzenia, czy zamówienie istnieje, zweryfikowania praw dostępu osoby, która zgłosiła żądanie anulowania, wyświetlenia zamówienia oraz żądania potwierdzenia anulowania. Przypadki testowe specyfikuje Analityk Testów, korzystając z sekwencji słów kluczowych (np. „Zaloguj”, „Wybierz_Zamówienie”, „Anuluj_Zamówienie”) oraz odpowiednich danych testowych.

Należy zwrócić uwagę na następujące zagadnienia:

- Im bardziej szczegółowe są słowa kluczowe, tym bardziej szczegółowe scenariusze można uwzględnić, jednak trudniej jest wówczas zarządzać językiem wysokiego poziomu.
- Umożliwienie analitykom testów specyfikowania działań niskiego poziomu („KliknijPrzycisk”, „WybierzZListy” itp.) pozwala obsłużyć w testach tego rodzaju większą liczbę różnych sytuacji. Jednak z uwagi na fakt, że działania te bezpośrednio wiążą się z graficznym interfejsem użytkownika, testy wymagają wykonania dodatkowych czynności pielęgnacyjnych po wprowadzeniu modyfikacji.
- Użycie zagregowanych słów kluczowych może z jednej strony uprościć projektowanie, z drugiej zaś może także utrudnić utrzymanie środowiska. Na przykład można zdefiniować sześć słów kluczowych, które razem służą do utworzenia rekordu. Jednakże utworzenie słowa kluczowego wysokiego poziomu, które wywołuje wszystkie sześć słów kluczowych, może nie być najbardziej efektywnym podejściem.
- Niezależnie od czasu poświęconego na analizę podczas tworzenia języka słów kluczowych może pojawić się konieczność dodania nowych lub zmiana istniejących słów kluczowych. Słowo kluczowe ma dwa odrębne aspekty, tj. stojącą za nim logikę biznesową i funkcję automatyzacji, która ją wywołuje. Należy zatem utworzyć proces, który obsługuje oba aspekty.

Automatyzacja testów oparta na słowach kluczowych może odznaczać się znacznie mniejszymi kosztami utrzymania niż koszt utrzymania innych rozwiązań. Początkowe wdrożenie może być bardziej kosztowne, ale jeśli projekt będzie trwał wystarczająco długo, będzie tańszy w ogólnym rozrachunku.

Sylabus Certyfikowany Tester – Inżynier Automatyzacji Testów [CT_TAE_SYL] zawiera dodatkowe informacje na temat modelowania procesów biznesowych na potrzeby automatyzacji.

6.2 Kategorie narzędzi testowych

W tym podrozdziale przedstawiono informacje na temat narzędzi, których może użyć Techniczny Analityk Testów, a które nie zostały omówione w sylabusie Certyfikowany Tester – Poziom podstawowy [CTFL_SYL].

Szczegółowe informacje na temat narzędzi można znaleźć w następujących sylabusach ISTQB®:

- Certyfikowany Tester – Tester Aplikacji Mobilnych [CT_MAT_SYL]
- Certyfikowany Tester – Tester Wydajności [CT_PT_SYL]
- Certyfikowany Tester – Testowanie Oparte Na Modelu [CT_MBT_SYL]
- Certyfikowany Tester – Inżynier Automatyzacji Testów [CT_TAE_SYL].

6.2.1 Narzędzia do posiewu usterek

Narzędzia do posiewu usterek modyfikują testowany kod (czasami z użyciem wstępnie zdefiniowanych algorytmów) w celu sprawdzenia pokrycia uzyskiwanego w określonych testach. Jeśli ta metoda jest stosowana w sposób systematyczny, umożliwia ocenę jakości testów (tj. ich zdolności do wykrycia wprowadzonych defektów) i, jeśli to niezbędne, poprawę jakości testów.

Narzędzia do posiewu usterek są zazwyczaj używane przez Technicznego Analityka Testów, ale mogą być również używane przez programistę podczas testowania nowo powstałego kodu.

6.2.2 Narzędzia do wstrzykiwania błędów

Narzędzia do wstrzykiwania błędów specjalnie wprowadzają do oprogramowania niepoprawne wartości wejściowe aby sprawdzić, czy poradzi sobie ono z obsługą defektu. Wartości te powodują negatywne warunki, które powinny spowodować uruchomienie (i przetestowanie) obsługi błędów. To zakłócenie normalnego przepływu wykonania kodu zwiększa również pokrycie kodu.

Narzędzia do wstrzykiwania błędów wykorzystywane są zwykle przez Technicznego Analityka Testów, ale mogą ich również używać programiści do testowania nowego kodu.

6.2.3 Narzędzia do testów wydajnościowych

Narzędzia do testów wydajnościowych realizują przede wszystkim następujące funkcje:

- generowanie obciążenia,
- pomiar, monitorowanie, wizualizacja i analiza odpowiedzi systemu na zadane obciążenie,
- udostępnianie informacji o zachowaniu zasobów w modułach systemu i modułach sieciowych.

Generowanie obciążenia odbywa się przez zaimplementowanie zdefiniowanego wcześniej profilu operacyjnego (patrz podrozdział 4.9.) w postaci skryptu. Skrypt może zostać zarejestrowany wstępnie dla jednego użytkownika (np. z wykorzystaniem narzędzia rejestrująco-odtwarzającego), a później zaimplementowany dla określonego profilu operacyjnego przy użyciu narzędzia do testów wydajnościowych. W implementacji konieczne jest wzięcie pod uwagę zróżnicowania danych w poszczególnych transakcjach (lub zbiorach transakcji).

Narzędzia do testów wydajnościowych generują obciążenie przez symulowanie dużej liczby użytkowników („wirtualnych") realizujących przypisane im profile operacyjne i generujących dane wejściowe o ustalonej wielkości. W porównaniu z poszczególnymi skryptami automatyzacji wykonywania testów skrypty testujące wydajność często odtwarzają interakcję użytkowników z systemem na poziomie protokołu komunikacyjnego, a nie przez symulowanie interakcji przy użyciu interfejsu GUI, jak to ma miejsce w przypadku standardowych skryptów automatyzacji. Zwykle pozwala to zmniejszyć liczbę odrębnych „sesji" niezbędnych w trakcie testowania. Niektóre narzędzia do generowania obciążenia sterują działaniem aplikacji również są pomocą interfejsu użytkownika, aby dokładniej zmierzyć czasy odpowiedzi systemu w warunkach obciążenia.

Narzędzia do testów wydajnościowych dokonują pomiarów w szerokim zakresie, które można przeanalizować w trakcie wykonywania testów lub po ich zakończeniu. Typowe metryki i raporty zawierają:

- liczbę symulowanych użytkowników,
- liczbę i typ transakcji generowanych przez symulowanych użytkowników oraz częstotliwość pojawiania się transakcji,
- czasy odpowiedzi na poszczególne żądania transakcji zgłaszane przez użytkowników,
- raporty i wykresy prezentujące obciążenie i czasy reakcji,
- raporty dotyczące wykorzystania zasobów (np. w funkcji czasu z podaniem wartości minimalnych i maksymalnych).

Głównymi czynnikami, które należy wziąć pod uwagę podczas wdrażania narzędzi do testów wydajnościowych, są:

- sprzęt i przepustowość sieci wymagane do wygenerowania obciążenia,
- kompatybilność narzędzia z protokołami komunikacyjnymi używanymi przez testowany system,
- elastyczność narzędzia pozwalająca na łatwe implementowanie różnych profili operacyjnych,
- urządzenia do monitorowania, analizy i raportowania.

Ze względu na dużą pracochłonność wytwarzania narzędzi do testów wydajnościowych nie są one zwykle opracowywane w ramach projektu, tylko nabywane. Może jednak okazać się zasadne opracowanie konkretnego narzędzia do testowania wydajności, jeśli z powodu ograniczeń technicznych nie da się skorzystać z gotowego produktu lub jeśli profil obciążenia i realizowane funkcje są proste w porównaniu z profilem obciążenia i obiektami dostarczanymi przez narzędzia komercyjne. Dodatkowe informacje na

temat narzędzi do testów wydajnościowych można znaleźć w sylabusie Certyfikowany Tester – Tester Wydajności [CT_PT_SYL].

6.2.4 Narzędzia do testowania stron internetowych

Dostępnych jest wiele wyspecjalizowanych narzędzi komercyjnych i narzędzi o otwartym kodzie (ang. *open source*), przeznaczonych do testowania stron internetowych. Na poniższej liście przedstawiono zastosowania niektórych najczęściej używanych narzędzi z tej kategorii:

- narzędzia do testowania hiperłączy służące do skanowania serwisu i wykrywania brakujących i niepoprawnych hiperłączy,
- narzędzia do sprawdzania kodu HTML i XML, weryfikujące zgodność kodu ze standardami HTML i XML dla poszczególnych stron tworzonych w serwisie internetowym,
- narzędzia do testowania wydajności służące do testowania reakcji serwera na podłączenie się dużej liczby użytkowników,
- lekkie narzędzia do wykonywania testów automatycznych, współpracujące z różnymi przeglądarkami,
- narzędzia do skanowania kodu serwera, poszukujące niepowiązanych („osieroconych") plików, wykorzystywanych wcześniej przez serwis internetowy,
- narzędzia do sprawdzania pisowni obsługujące HTML,
- narzędzia do sprawdzania arkuszy CSS,
- narzędzia do sprawdzania przypadków naruszenia norm, np. standardów dostępności (Section 508 w Stanach Zjednoczonych i M/376 w Europie),
- narzędzia do wykrywania różnych rodzajów problemów dotyczących zabezpieczeń.

Poniższe serwisy są dobrymi źródłami narzędzi typu *open source* do testowania stron internetowych:

- World Wide Web Consortium (W3C) [Web-3] — organizacja odpowiedzialna za jego zawartość ustala standardy internetowe i udostępnia różnorodne narzędzia do wyszukiwania niezgodności z tymi standardami,
- Web Hypertext Application Technology Working Group (WHATWG) [Web-5] — organizacja, która określa standardy języka HTML i oferuje narzędzie przeznaczone do sprawdzania poprawności kodu HTML [Web-6].

Niektóre narzędzia zawierające roboty indeksujące (ang. *web spider*) mogą również dostarczać informacje na temat wielkości stron i czasu niezbędnego do ich pobrania, a także dostępności poszczególnych stron (i np. występowaniu błędu 404 protokołu HTTP). To przydatne informacje dla programistów, administratorów stron i testerów.

Analitycy testów i techniczni analitycy testów używają takich narzędzi głównie w trakcie testów systemowych.

6.2.5 Narzędzia wspomagające testowanie oparte na modelu

Testowanie oparte na modelu (ang. *model-based testing*, *MBT*) to technika, w której do opisu pożądanego zachowania systemu sterowanego oprogramowaniem w czasie wykonywania używany jest pewien model, na przykład diagram przejść między stanami. Komercyjne narzędzia MBT (patrz [Utting07]) często udostępniają mechanizm umożliwiający użytkownikom „wykonanie" modelu. Interesujące wątki wykonywanego modelu można zapisać i wykorzystać jako przypadki testowe. Inne wykonywalne modele, np. sieci Petriego i diagramy przejść między stanami, również wspierają ten rodzaj testowania (MBT).

Modeli (i narzędzi) MBT można użyć do wygenerowanie dużych zbiorów odrębnych wątków wykonywanych. Narzędzia tego typu pozwalają również ograniczyć bardzo dużą liczbę możliwych ścieżek, które mogą zostać wygenerowane w ramach modelu. Testowanie przy użyciu tych narzędzi daje możliwość uzyskania innej perspektywy na testowane oprogramowanie. Można w ten sposób wykryć defekty, które nie zostały zauważone w trakcie testowania funkcjonalnego.

Dodatkowe informacje na temat narzędzi do testów opartych na modelu można znaleźć w sylabusie Certyfikowany Tester – Testowanie Oparte Na Modelu [CT_MBT_SYL].

6.2.6 Narzędzia do testowania modułowego i budowania wersji

Narzędzia do testowania modułowego i budowania wersji są przeznaczone dla programistów, jednak w wielu przypadkach są używane i utrzymywane przez technicznych analityków testów, zwłaszcza w kontekście projektów prowadzonych z zastosowaniem metodyk zwinnych.

Narzędzia do testowania modułowego są często związane z językiem używanym do programowania danego komponentu. Na przykład jeśli językiem programowania jest Java, do automatyzacji testów modułowych można użyć środowiska JUnit. Z wieloma innymi językami związane są specjalne narzędzia testowe; są one ogólnie nazywane xUnit. W środowiskach tych obiekty testowe generowane są dla każdej utworzonej klasy, co pozwala uprościć zadania programistów podczas automatyzacji testów modułowych.

Niektóre narzędzia do automatyzacji budowania wersji dają możliwość automatycznego uruchomienia procesu budowania nowej wersji po zmodyfikowaniu modułu. Po zakończeniu takiej operacji inne narzędzia automatycznie uruchamiają testy modułowe. Taki poziom automatyzacji procesu budowania wersji występuje zwykle w środowiskach ciągłej integracji.

Poprawnie skonfigurowany zestaw narzędzi tego typu może mieć istotny wpływ na poprawę jakości wersji przekazywanych do testowania. Jeśli zmiany wprowadzone przez programistę wiążą się z wprowadzeniem defektów regresji w danej wersji, zwykle będzie wiązać się to z niepowodzeniem wykonania niektórych testów automatycznych. Zanim wersja zostanie przekazana do środowiska testowego, programista podejmuje próbę pośredniego badania przyczyny awarii.

6.2.7 Narzędzia wspomagające testowanie aplikacji mobilnych

Narzędziami często stosowanymi do testowania aplikacji mobilnych są symulatory i emulatory.

6.2.7.1. Symulatory

Symulator urządzenia mobilnego służy do imitowania środowiska wykonawczego danej platformy mobilnej. Testowane aplikacje są kompilowane do specjalnej wersji, która działa na symulatorze, a nie na rzeczywistym urządzeniu. Symulatory są używane jako zamienniki prawdziwych urządzeń w testach, ale zazwyczaj ich zastosowanie ogranicza się do wstępnych testów funkcjonalnych i symulowania wielu wirtualnych użytkowników w testach obciążeniowych. Symulatory są stosunkowo proste (w porównaniu do emulatorów) i mogą wykonać testy szybciej niż emulator. Należy jednak pamiętać, że testowana na symulatorze aplikacja różni się od produktu, który ma trafić do dystrybucji.

6.2.7.2. Emulatory

Emulator urządzenia mobilnego służy do imitowania działania sprzętu i używa tego samego środowiska wykonawczego, z którego korzystają urządzenia fizyczne. Aplikacje skompilowane w celu wdrożenia i przetestowania na emulatorze mogą również działać na docelowych urządzeniach.

Emulator nie jest jednak w stanie w pełni zastąpić urządzenia mobilnego, którego działanie ma odtwarzać, ponieważ może zachowywać się w inny sposób. Ponadto może nie obsługiwać niektórych funkcji, np. ekranu dotykowego (także wielopunktowego) czy akcelerometru. Wynika to między innymi z ograniczeń występujących na platformie, na której działa emulator.

6.2.7.3 Wspólne aspekty zastosowań

Symulatory i emulatory są często używane w celu zmniejszenia kosztów środowisk testowych poprzez zastąpienie rzeczywistych urządzeń. Symulatory i emulatory są przydatne we wczesnych fazach cyklu wytwórczego, ponieważ zazwyczaj można je zintegrować ze środowiskami programistycznymi, a przy tym pozwalają szybko wdrażać, testować i monitorować aplikacje. Aby można było użyć emulatora lub symulatora, należy go uruchomić i zainstalować na nim odpowiednią aplikację – taką, jak na rzeczywistym urządzeniu. Każde środowisko programistyczne mobilnego systemu operacyjnego zawiera zwykle własny emulator lub symulator. Dostępne są także emulatory i symulatory pochodzące z innych źródeł.

Emulatory i symulatory zwykle umożliwiają ustawienie wartości różnych parametrów użytkowych. Takie ustawienia mogą obejmować m.in. emulację działania sieci z różnymi szybkościami i różnymi poziomami sygnału, utratę pakietów, zmianę orientacji, generowanie przerwań i wykorzystanie danych lokalizacyjnych GPS. Niektóre ustawienia mogą okazać się bardzo przydatne, ponieważ ich odtworzenie za pomocą fizycznych urządzeń może być trudne lub kosztowne (np. lokalizacja GPS czy poziom sygnału).

Dodatkowe informacje na ten temat można znaleźć w sylabusie Certyfikowany Tester – Tester Aplikacji Mobilnych [CT_MAT_SYL].

7. Dokumenty pomocnicze

7.1 Normy i standardy

W odpowiednich rozdziałach niniejszego dokumentu odwołano się do następujących standardów:

- [DO-178C]: Software Considerations in Airborne Systems and Equipment Certification, RTCA 2011. (Rozdział 2)
- [ISO 9126] ISO/IEC 9126-1:2001, Software Engineering — Software Product Quality (Rozdział 4 oraz Załącznik A)
- [ISO 25010] ISO/IEC 25010:2011, Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) System and Software Quality Models (Rozdziały 1 i 4 oraz Załącznik A)
- [ISO 29119] ISO/IEC/IEEE 29119-4 Software and Systems Engineering — Software Testing Part 4: Test Techniques. (Rozdział 2)
- [ISO 42010] ISO/IEC/IEEE 42010:2011 Systems and Software Engineering — Architecture description (Rozdział 5)
- [IEC 61508] IEC 61508-5 (2010) Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, Part 5: Examples of methods for the determination of safety integrity levels (Rozdział 2)
- [ISO 26262] ISO 26262-1:2018, Road vehicles — Functional safety, części 1 do 12. (Rozdział 2)
- [IEC 62443-3-2] IEC 62443-3-2:2020, Security for industrial automation and control systems, część 3-2:Security risk assessment for system design (Rozdział 4).

7.2 Dokumenty ISTQB®

- [ISTQB_AL_TTA_OVIEW] Certyfikowany tester ISTQB ® Sylabus poziomu zaawansowanego Techniczny Analityk Testów (TTA)Omówienie sylabusa, wersja 4.0
- [CT_SEC_SYL] Sylabus Certyfikowany Tester – Tester Zabezpieczeń, wersja 1.0
- [CT_TAE_SYL] Sylabus Certyfikowany Tester – Inżynier Automatyzacji Testów, wersja 1.0
- [CTFL_SYL] Sylabus Certyfikowany Tester – Poziom Podstawowy, wersja 3.1.
- [CT_PT_SYL] Sylabus Certyfikowany Tester – Tester Wydajności, wersja 1.0
- [CT_MBT_SYL] Sylabus Certyfikowany Tester – Testowanie Oparte Na Modelu, wersja 1.0
- [CT_TM_SYL] Sylabus Certyfikowany Tester – Poziom Zaawansowany – Kierownik Testów, wersja 2.0
- [CT_MAT_SYL] Sylabus Certyfikowany Tester – Tester Aplikacji Mobilnych, wersja 1.0
- [ISTQB_GLOSSARY] Słownik terminów testowych ISTQB, https://glossary.istqb.org/pl/search
- [CT_AuT_SYL] Sylabus Certyfikowany Tester – Tester Oprogramowania Motoryzacyjnego, wersja 1.0

7.3 Książki i artykuły

[Andrist20] Björn Andrist and Viktor Sehr, C++ High Performance: Master the art of optimizing the functioning of your C++ code, 2nd Edition, Packt Publishing, 2020
[Beizer90] Boris Beizer, „Software Testing Techniques, Second Edition", International Thomson Computer Press, 1990, ISBN 1-8503-2880-3
[Buwalda01] Hans Buwalda, „Integrated Test Design and Automation", Addison-Wesley Longman, 2001, ISBN 0-201-73725-6
[Kaner02] Cem Kaner, James Bach, Bret Pettichord; „Lessons Learned in Software Testing"; Wiley, 2002, ISBN: 0-471-08112-4
[McCabe76] Thomas J. McCabe, „A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, 1976 r. PP 308-320
[Roman18] Adam Roman, "Testowanie i jakość oprogramowania. Modele, techniki, narzędzia", Polskie Wydawnictwo Naukowe PWN, 2018, ISBN 978-83-01-19644-8
[Utting07] Mark Utting, Bruno Legeard, „Practical Model-Based Testing. A Tools Approach", Morgan-Kaufmann, 2007, ISBN: 978-0-12-372501-1
[Whittaker04] James Whittaker i Herbert Thompson, „How to Break Software Security", Pearson / Addison-Wesley, 2004, ISBN 0-321-19433-0
[Wiegers02] Karl Wiegers, „Peer Reviews in Software: A Practical Guide", Addison-Wesley, 2002, ISBN 0-201-73485-0

7.4 Inne źródła

Wymienione poniżej źródła wskazują informacje dostępne w Internecie. Odwołania zostały sprawdzone w momencie publikacji niniejszego sylabusa poziomu zaawansowanego. ISTQB® nie ponosi jednak odpowiedzialności za ich ewentualną późniejszą niedostępność.

[Web-1] http://www.nist.gov NIST National Institute of Standards and Technology,
[Web-2] http://www.codeproject.com/KB/architecture/SWArchitectureReview.aspx
[Web-3] http://www.W3C.org
[Web-4] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
[Web-5] https://whatwg.org
[Web-6] https://validator.w3.org/
[Web-7] https://dl.acm.org/doi/abs/10.1145/3340433.3342822

Rozdział 2.: [Web-7]
Rozdział 4.: [Web-1] [Web-4]
Rozdział 5.: [Web-2]
Rozdział 6.: [Web-3] [Web-5] [Web-6]

8. Załącznik A: Przegląd charakterystyk jakościowych

W poniższej tabeli przedstawiono porównanie charakterystyk jakościowych opisanych w nieaktualnym już standardzie ISO 9126-1 (używanych w wersji 2012 sylabusa dla Technicznego Analityka Testów) z charakterystykami pochodzącymi z nowszego standardu ISO 25010 [ISO 25010] (używanymi w najnowszej wersji sylabusa). Należy pamiętać, że przydatność funkcjonalna i użyteczność są omówione w sylabusie dla Analityka Testów.

ISO/IEC 25010	ISO/IEC 9126-1	Uwagi
Funkcjonalność (przydatność funkcjonalna)	**Funkcjonalność**	Nowa nazwa jest bardziej precyzyjna i pozwala uniknąć pomyłek związanych z innymi znaczeniami terminu "funkcjonalność".
Kompletność funkcjonalna		Pokrycie określonych potrzeb
Poprawność funkcjonalna	Dokładność	Bardziej ogólne niż dokładność
Adekwatność funkcjonalna	Dopasowanie	Pokrycie domniemanych potrzeb
	Współdziałanie	Przeniesione do Kompatybilności
	Zabezpieczenia	Teraz osobna charakterystyka
Wydajność	**Efektywność**	Zmieniona nazwa w celu uniknięcia konfliktu z definicją efektywności w ISO/IEC 25062
Zachowanie w czasie	Zachowanie w czasie	
Zużycie zasobów	Zużycie zasobów	
Pojemność		Nowa podcharakterystyka
Kompatybilność		Nowa charakterystyka
Współistnienie	Współistnienie	Charakterystyka przeniesiona z Przenaszalności
Współdziałanie		Charakterystyka przeniesiona z funkcjonalności (Analityk Testów)
Użyteczność		Wyraźnie widoczna kwestia jakości
Stosowność	Zrozumiałość	Nowa nazwa jest bardziej precyzyjna
Łatwość nauki	Łatwość nauki	
Łatwość obsługi	Łatwość obsługi	
Ochrona przed błędami użytkownika		Nowa podcharakterystyka
Estetyka interfejsu użytkownika	Atrakcyjność	Nowa nazwa jest bardziej precyzyjna
Dostępność		Nowa podcharakterystyka
Niezawodność	**Niezawodność**	
Dojrzałość	Dojrzałość	
Osiągalność		Nowa podcharakterystyka
Tolerowanie usterek	Tolerowanie usterek	
Odtwarzalność	Odtwarzalność	
Zabezpieczenia	**Zabezpieczenia**	Brak poprzednich podcharakterystyk
Poufność		Brak poprzednich podcharakterystyk
Integralność		Brak poprzednich podcharakterystyk
Niezaprzeczalność		Brak poprzednich podcharakterystyk
Rozliczalność		Brak poprzednich podcharakterystyk
Autentykacja		Brak poprzednich podcharakterystyk

Utrzymywalność	**Utrzymywalność**	
Modułowość		Nowa podcharakterystyka
Możliwość ponownego użycia		Nowa podcharakterystyka
Analizowalność	Analizowalność	
Modyfikowalność	Stabilność	Bardziej precyzyjna nazwa łącząca modyfikowalność i stabilność
Testowalność	Testowalność	
Przenaszalność	**Przenaszalność**	
Zdolność adaptacyjna	Zdolność adaptacyjna	
Instalowalność	Instalowalność	
	Współistnienie	Charakterystyka przeniesiona do Kompatybilności
Zastępowalność	Zastępowalność	

9. Indeks