

# **Certyfikowany tester**

## **Plan poziomu podstawowego**

**Wersja 1.0**

---

Stowarzyszenie Jakości Systemów Informatycznych

---

International Software Testing Qualifications Board

---

Wszelkie prawa dla wersji angielskiej zastrzeżone dla © International Software Testing Qualifications Board (dalej nazywane ISTQB).

Grupa Robocza ISTQB, plan Poziomu Podstawowego : Thomas Müller (przewodniczący), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson and Erik van Veendendal **2004-2005**.

Prawa autorskie zastrzeżone © Stowarzyszenie Jakości Systemów Informatycznych (SJSI).

Tłumaczenie z języka angielskiego oraz udział w przeglądach: Bogdan Bereza-Jarociński, Wojciech Jaszcz, Helena Klitenik, Joanna Nowakowska, Jan Sabak, Anna Seredyn, Lucjan Stapp, Piotr Ślęzak, Łukasz Żebrowski.

Autorzy, tłumacze, ISTQB oraz SJSI określają następujące warunki korzystania z Planu:

- 1) Osoby oraz firmy szkoleniowe mają prawo korzystać z planu jako podstawy do materiałów szkoleniowych pod warunkiem podania źródła praw autorskich oraz własności planu. Powoływanie się na niniejszy plan w materiałach reklamowych i promocyjnych dozwolone jest dopiero po oficjalnym rozpoczęciu procedury ubiegania się o akredytację w ISTQB lub w Radzie Krajowej (*National Board*) uznawanej przez ISTQB.
- 2) Osoby oraz firmy i zrzeszenia mają prawo korzystać z planu jako podstawy do artykułów, książek oraz innych materiałów pod warunkiem podania źródła praw autorskich oraz własności planu.
- 3) Każda uznawana przez ISTQB Rada Krajowa może wykonać tłumaczenie niniejszego planu oraz udzielać zezwolenia na korzystanie z całości lub części tłumaczenia innym stronom.

## Historia zmian

Wersja	Data	Uwagi
0.1	15.07.2005	Wzorzec dokumentu.
0.1.1	01.08.2005	Bogdan Bereza-Jarociński, częściowo rozdział 6.
0.1.2	16.08.2005	Anna Seredyn: fragmenty rozdz. 1 (kosmetyczne zmiany BB). Bogdan Bereza-Jarociński: dalsze tłumaczenie rozdziału 6-ego. Helena Klitenik: załącznik A (kosmetyczne zmiany BB).
0.1.3	30.09.2005	Dodany rozdział 3. (Helena Klitenik), rozdział 4. (Helena Klitenik) oraz Wstęp (Helena Klitenik).
0.2	02.01.2006	Do przeglądu (Bogdan Bereza-Jarociński)
0.2.1	04.01.2006	Uzupełnienia: załączniki B, C i D od Heleny Klitenik
0.9	06.02.2006	Wersja dostarczona do SJSI
0.91	10.02.2006	Wersja do ostatecznego przeglądu.
0.92	17.02.2006	Scalona wersja po przeglądzie, który nie okazał się ostateczny
0.93	28.07.2006	Poprawki do wersji ostatecznego przeglądu
0.94	11.09.2006	Wersja alfa
0.95	18.09.2006	Wersja beta. Dodanie indeksu.
1.0	20.10.2006	Dostarczona zarządowi SJSI

## Spis treści

Historia zmian.....	3
Spis treści.....	4
Podziękowania .....	7
Wstęp do planu.....	8
1. Podstawy testowania (K2), 155 minut.....	10
1.1. Czemu testowanie jest niezbędne (K2), 20 minut.....	11
1.1.1. Otoczenie systemów informatycznych (K1).....	11
1.1.2. Przyczyny defektów oprogramowania (K2) .....	11
1.1.3. Rola testowania w procesach tworzenia, utrzymania i użytkowania oprogramowania (K2).....	11
1.1.4. Testowanie i jakość (K2).....	11
1.1.5. Kiedy zakończyć testowanie (K2).....	12
1.2. Co to jest testowanie (K2), 30 minut.....	13
1.3. Ogólne zasady testowania (K2), 35 minut .....	14
1.4. Prosty proces testowy (K1), 35 minut.....	15
1.4.1. Planowanie i nadzór nad testowaniem (K1) .....	15
1.4.2. Analiza i projektowanie testów (K1).....	16
1.4.3. Implementacja i wykonanie testów (K1) .....	16
1.4.4. Ocena kryteriów zakończenia oraz raportowanie testów (K1) .....	16
1.4.5. Czynności na zakończenie testowania (K1) .....	17
1.5. Psychologia testowania (K2), 35 minut.....	18
2. Testowanie w cyklu życia oprogramowania (K2), 135 minut.....	20
2.1. Modele wytwarzania oprogramowania (K2), 20 minut .....	21
2.1.1. Model V (K2).....	21
2.1.2. Iteracyjne modele wytwarzania (K2).....	21
2.1.3. Testowanie w cyklu życia (K2) .....	21
2.2. Poziomy testów (K2), 60 minut.....	23
2.2.1. Testowanie modułowe (K2).....	23
2.2.2. Testowanie integracyjne (K2).....	23
2.2.3. Testowanie systemowe (K2).....	24
2.2.4. Testowanie akceptacyjne (K2) .....	25
2.3. Typy i cele testów (K2), 40 minut .....	26
2.3.1. Testowanie funkcji (testowanie funkcjonalne) (K2).....	26
2.3.2. Testowanie właściwości (testowanie нефunkcjonalne) (K2).....	26
2.3.3. Testowanie struktury/architektury systemu (testowanie strukturalne) (K2).....	27
2.3.4. Testowanie związane ze zmianami (testowanie potwierdzające i regresywne) (K2) .....	27
2.4. Testowanie w fazie utrzymania (K2), 15 minut .....	28
3. Statyczne techniki testowania (K2), 60 minut .....	29
3.1. Przeglądy i proces testowy(K2), 15 minut.....	30
3.2. Proces przeglądu (K2), 25 minut .....	31
3.2.1. Fazy formalnego przeglądu (K1).....	31
3.2.2. Role i odpowiedzialności (K1) .....	31
3.2.3. Rodzaje przeglądów (K2).....	32

3.2.4.	Czynniki powodzenia przeglądów (K2).....	33
3.3.	Analiza statyczna przy pomocy narzędzi (K2), 20minut.....	34
4.	Techniki projektowania testów (K3), 255 minut.....	35
4.1.	Identyfikowanie warunków testowych i projektowanie przypadków testowych (K3), 15 minut .....	37
4.2.	Kategorie technik projektowania testów (K2), 15 minut.....	38
4.3.	Techniki na podstawie specyfikacji lub czarnoskrzynkowe (K3), 120 minut .....	39
4.3.1.	Podział na klasy równoważności (K3) .....	39
4.3.2.	Analiza wartości brzegowych (K3) .....	39
4.3.3.	Testowanie w oparciu o tablicę decyzyjną (K3) .....	39
4.3.4.	Testowanie przejść pomiędzy stanami (K3) .....	40
4.3.5.	Testowanie w oparciu o przypadki użycia (K2).....	40
4.4.	Techniki na podstawie struktury lub białoskrzynkowe (K3), 60 minut.....	41
4.4.1.	Testowanie instrukcji i pokrycie (K3) .....	41
4.4.2.	Testowanie decyzyjne i pokrycie (K3) .....	41
4.4.3.	Inne techniki na podstawie struktury (K1).....	41
4.5.	Techniki oparte na doświadczeniu (K2), 30 minut .....	42
4.6.	Wybór technik testowych (K2), 15 minut.....	43
5.	Zarządzanie testowaniem (K3), 180 minut.....	44
5.1.	Organizacja testowania (K2), 30 minut.....	46
5.1.1.	Organizacja i niezależność testowania (K2) .....	46
5.1.2.	Zadania kierownika testów i testera (K1).....	46
5.2.	Planowanie testowania (K2), 50 minut .....	48
5.2.1.	Planowanie testowania (K2).....	48
5.2.2.	Czynności wykonywane podczas planowania testów (K2) .....	48
5.2.3.	Kryteria wyjścia (K2) .....	48
5.2.4.	Oszacowanie wysiłku testowego (K2) .....	49
5.2.5.	Sposoby podejścia do testowania (strategie testowe) (K2).....	49
5.3.	Monitorowanie przebiegu i nadzór testowania (K2), 20 minut.....	51
5.3.1.	Monitorowanie postępu testów (K1) .....	51
5.3.2.	Raportowanie testów (K2).....	51
5.3.3.	Nadzór nad testowaniem (K2).....	51
5.4.	Zarządzanie konfiguracją (K2), 10 minut .....	53
5.5.	Ryzyko a testowanie (K2), 30 minut .....	54
5.5.1.	Ryzyko projektowe (K1, K2).....	54
5.5.2.	Ryzyko związane z produktem (K2) .....	54
5.6.	Zarządzanie incydentami (K3), 40 minut .....	56
6.	Testowanie wspierane narzędziami (K2), 80 minut .....	58
6.1.	Rodzaje narzędzi testowych (K2), 45 minut.....	59
6.1.1.	Klasyfikacja narzędzi testowych.....	59
6.1.2.	Zarządzanie testowaniem wspierane narzędziami (K1).....	59
6.1.3.	Testowanie statyczne wspierane narzędziami (K1).....	61
6.1.4.	Tworzenie specyfikacji testów wspierane narzędziami .....	61
6.1.5.	Wykonywanie testów wspierane narzędziami .....	62
6.1.6.	Testowanie wydajności i monitorowanie wspierane narzędziami (K1) .....	63

6.1.7.	Narzędzia wspierające testowanie w określonych dziedzinach .....	63
6.1.8.	Wykorzystanie innych narzędzi (K1) .....	64
6.2.	Skuteczne zastosowanie narzędzi: możliwe korzyści i zagrożenia (K2), 20 minut .....	65
6.2.1.	Możliwe korzyści i zagrożenia wynikające ze stosowania narzędzi wspierających testowanie (dla wszystkich rodzajów narzędzi) (K2).....	65
6.2.2.	Zagadnienia specjalne dla niektórych rodzajów narzędzi (K1).....	65
6.3.	Wdrożenie narzędzia w organizacji (K1), 15 minut .....	67
7.	Referencje .....	69
Załącznik A – Pochodzenie planu.....		70
Załącznik B – Cel nauki i poziomy wiedzy .....		72
Załącznik C – Zasady dotyczące Planu poziomu podstawowego ISTQB / SJSI .....		73
Załącznik D – Informacja dla dostawców szkoleń.....		75
Indeks.....		76

## Podziękowania

Angielska wersja tego dokumentu została napisana przez Grupę Roboczą ISTQB w składzie: Thomas Müller (przewodniczący), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson oraz Erik van Veendendal. Grupa Robocza wyraża podziękowania przedstawicielom Rad Krajowych za udział w przeglądach planu.

Szczególne podziękowania skierowane są do (Austria) Anastasios Kyriakopoulos, (Dania) Klaus Olsen, Christie Rosenbeck-Larsen, (Niemcy) Matthias Daigl, Uwe Hehn, Tilo Linz, Horst Pohlmann, Ina Schieferdecker, Sabine Uhde, Stephanie Ulrich, (Indie) Vipul Kocher, (Izrael) Shmuel Knishinsky, Ester Zabar, (Szwecja) Anders Claesson, Mattias Nordin, Ingvar Nordström, Stefan Ohlsson, Kenneth Osbjør, Ingela Skytte, Klaus Zeuge, (Szwajcaria) Armin Born, Sandra Harries, Silvio Moser, Reto Müller, Joerg Pietzsch, (Wielka Brytania) Aran Ebbett, Isabel Evans, Julie Gardiner, Andrew Goslin, Brian Hambling, James Lyndsay, Helen Moore, Peter Morgan, Trevor Newton, Angelina Samaroo, Shane Saunders, Mike Smith, Richard Taylor, Neil Thompson, Pete Williams, (USA) Dale Perry.

Tłumaczenie na język polski zostało wykonane przez Grupę Roboczą SJSI w składzie: Bogdan Bereza-Jarociński, Wojciech Jaszcz, Helena Klitenik, Joanna Nowakowska, Jan Sabak, Anna Seredyn, Lucjan Stapp, Piotr Ślęzak, Łukasz Żebrowski.

## Wstęp do planu

### **Cel dokumentu**

Plan ten stanowi podstawę do Międzynarodowej Kwalifikacji w Testowaniu Oprogramowania na poziomie podstawowym. ISTQB (*The International Software Testing Qualifications Board*) zapewnia narodowym grupom egzaminacyjnym akredytację dostawców szkoleń oraz posiadanie pytań egzaminacyjnych w ich lokalnym języku. Dostawcy szkoleń przygotowują materiał kursu i określają odpowiednie metody nauczania do akredytacji, a plan pomaga kandydatom w przygotowaniu się do egzaminu.

Informację o historii i pochodzeniu planu można znaleźć w Załączniku A.

### **Podstawowy Poziom Certyfikowanego Testera w Testowaniu Oprogramowania**

Kwalifikacja na Poziomie Podstawowym kierowana jest do każdego zaangażowanego w testowanie oprogramowania. Zalicza się tu takie osoby jak testerzy, analitycy testów, inżynierowie testów, konsultanci testów, menedżerowie testów, testerzy akceptacji użytkownika i programiści. Kwalifikacja na Poziomie Podstawowym odpowiednia jest również dla każdego, kto potrzebuje podstawowej wiedzy w dziedzinie testowania oprogramowania. Dotyczy to kierowników projektu, kierowników jakości, kierowników oprogramowania, analityków biznesowych, dyrektorów IT i konsultantów zarządu. Posiadacze Certyfikatu Podstawowego będą mieli możliwość przejścia na wyższy poziom kwalifikacji w testowaniu oprogramowania.

### **Cele uczenia się/poziom wiedzy**

Dla każdej sekcji planu podane są następujące poziomy poznawcze:

K1: zapamiętaj, poznaj, wymień;

K2: zrozum, wyjaśnij, podaj przyczyny, porównaj, sklasyfikuj, streść;

K3: zastosuj.

Dalsze szczegóły i przykłady celów uczenia się podano w Załączniku B.

Wszystkie pojęcia wymienione w „Terminologii”, zaraz poniżej nagłówków rozdziałów, powinny być zapamiętane, nawet, jeśli nie wspomniano o tym wyraźnie, w celach uczenia się.

### **Egzamin**

Egzamin na Certyfikat Podstawowy będzie oparty na niniejszym konspekcie. Odpowiedzi na pytania egzaminacyjne mogą wymagać skorzystania z materiału bazującego na więcej niż jednym rozdziale niniejszego planu. W egzaminie uwzględnione są wszystkie rozdziały planu.

Format egzaminu to test wielokrotnego wyboru.

Egzaminy można zdawać podczas akredytowanego kursu szkoleniowego lub przystępować do nich niezależnie (np. w centrum egzaminacyjnym).

### **Akredytacja**

Dostawcy szkoleń, których materiał kursowy odpowiada planowi, mogą być akredytowani przez narodową organizację uznawaną przez ISTQB. Wytyczne odnośnie akredytacji powinno się uzyskać od organizacji lub ciała, które dokonuje akredytacji. Kurs akredytowany uznawany jest jako odpowiadający temu planowi. Zezwala się, aby zdawać egzamin ISTQB jako część kursu.

Dalsze wskazówki odnośnie dostawców szkoleń podano w Załączniku D.

## **Poziom uszczegółowienia**

Poziom uszczegółowienia w konspekcie pozwala na spójne międzynarodowo nauczanie oraz egzamin. Aby osiągnąć ten cel plan zawiera:

- Ogólne cele opisujące intencję poziomu podstawowego,
- Listę informacji do nauczania, wliczając w to opis i odniesienia do dodatkowych źródeł, jeśli są wymagane,
- Cele uczenia się dla każdej dziedziny wiedzy, opisujące poznawczy rezultat uczenia się i zestaw pojęciowy do osiągnięcia,
- Listę pojęć, które studenci muszą zrozumieć i umieć wymienić,
- Opis kluczowych koncepcji nauczania, wliczając w to źródła takie jak przyjęta literatura lub standardy.

Treść planu nie jest opisem całego obszaru wiedzy z testowania oprogramowania; odzwierciedla poziom szczegółowości dla kursów szkoleniowych z poziomu podstawowego.

## **Organizacja planu**

Plan zawiera sześć głównych rozdziałów. Nagłówek górnego poziomu pokazuje poziomy wiedzy zawarte wewnątrz rozdziału i określa jego czas trwania. Na przykład:

2. Testowanie w cyklu życia oprogramowania (K2), 135 minut
--

pokazuje, że rozdział 2 ma poziom wiedzy K1 (zakłada się zawieranie niższych, gdy pokazany jest wyższy poziom) i K2 (ale nie K3) i zaplanowano 135 minut na nauczanie materiału w nim zawartego.

Wewnątrz każdego rozdziału jest wiele sekcji. Każda sekcja ma również cele uczenia się i ilość wymaganego czasu. Podsekcje, dla których nie podano czasu, wliczają się do czasu dla sekcji.

## **1. Podstawy testowania (K2), 155 minut**

***Celem rozdziału „Podstawy testowania” jest nauczenie, w poszczególnych paragrafach:***

### **1.1 Dlaczego testowanie jest niezbędne (K2)**

- Przedstawiania, z przykładami, w jaki sposób błąd w oprogramowaniu może wyrządzić szkodę osobie, środowisku lub firmie. (K2)
- Różnic pomiędzy przyczyną błędu a jej efektem. (K2)
- Uzasadnienia, opierając się na przykładach, dlaczego testowanie jest niezbędne. (K2)
- Pokazania, dlaczego testowanie jest częścią zapewnienia jakości i podania przykładów, w jaki sposób testowanie pomaga osiągnąć wyższą jakość. (K2)
- Definicji: pomyłka, defekt, awaria oraz synonimy błąd i pluskwa. (K1)

### **1.2 Co to jest testowanie (K2)**

- Podstawowych zadań procesu testowania.(K1)
- Opisu celów testowania w procesach tworzenia, utrzymania i użytkowania oprogramowania, jako metody znajdowania błędów, dostarczenia informacji o jakości oraz zapobieganiu awariom oprogramowania. (K2)

### **1.3 Podstawowe zasady testowania (K2)**

- Podstawowych zasad testowania. (K2)

### **1.4 Podstawowy proces testowy (K1)**

- Podstawowych etapów testów od planowania do zakończenia testów i głównych zadań każdego etapu testów. (K1)

### **1.5 Psychologia testowania (K2)**

- Wpływu czynników psychologicznych na sukces testów poprzez (K1):
  - Jasno zdefiniowane cele;
  - Równowagę między testami programistów a testami niezależnymi;
  - Znaczenia asertywnej komunikacji i informacji o błędach.
- Różnicy w podejściu testera i programisty. (K2)

## 1.1. Czemu testowanie jest niezbędne (K2), 20 minut

### Terminologia

Pluskwa, defekt, pomyłka, awaria, usterka, jakość, ryzyko, oprogramowanie, testowanie.

#### 1.1.1. Otoczenie systemów informatycznych (K1)

Systemy informatyczne odgrywają coraz większą rolę w życiu, począwszy od rozwiązań dla biznesu (np. sektor bankowy) aż do urządzeń dla konsumenta (np. samochody). Większość ludzi zetknęła się z oprogramowaniem, które nie działało tak jak powinno. Oprogramowanie takie może wywołać wiele problemów – stratę pieniędzy, czasu i reputacji firmy, może nawet spowodować obrażenia lub śmierć.

#### 1.1.2. Przyczyny defektów oprogramowania (K2)

Człowiek popełnia błąd (pomyłkę), której skutkiem jest defekt (usterka, pluskwa) w kodzie, oprogramowaniu, systemie lub dokumencie. Jeśli kod z defektem zostanie wykonany, system nie zrobi tego, co powinien (lub zrobi coś, czego nie powinien robić), wywołując awarię. Defekty w oprogramowaniu, systemach lub dokumentach mogą, lecz nie muszą skutkować awariami.

Defekty istnieją, ponieważ ludzie są omylni, działają pod presją czasu, w sytuacji skomplikowanego kodu, skomplikowanej infrastruktury, zmieniających się technologii i wielu interakcji wewnątrz systemu.

Awarie mogą być również wywołane przez czynniki środowiska: promieniowanie, magnetyzm, pole elektryczne, skażenie. Wszystko to może powodować awarie w oprogramowaniu wbudowanym lub wpływać na działanie oprogramowania przez zmianę warunków działania sprzętu.

#### 1.1.3. Rola testowania w procesach tworzenia, utrzymania i użytkowania oprogramowania (K2)

Rygorystyczne testowanie systemów i dokumentacji może zredukować ryzyko wystąpienia awarii w środowisku produkcyjnym i przyczynić się do osiągnięcia wysokiej jakości systemu. Konieczne jest, aby znalezione defekty były poprawione przed dopuszczeniem systemu do działania w środowisku produkcyjnym.

Testowanie oprogramowania może być również wymagane przez zapisy kontraktowe, wymogi prawne lub standardy przemysłowe.

#### 1.1.4. Testowanie i jakość (K2)

Przy pomocy testowania można zmierzyć jakość oprogramowania rozumianą jako liczba znalezionych błędów, zarówno w stosunku do wymagań funkcjonalnych jak i нефункциональных oraz cech systemu (niezawodność, użyteczność, efektywność i utrzymywalność). Więcej informacji o testach нефункциональных znajduje się w rozdziale 2. Więcej informacji o cechach oprogramowania można znaleźć w normie *Software Engineering – Software Product Quality* (ISO 9126).

Testowanie może podnosić zaufanie co do jakości oprogramowania, jeżeli znaleziono mało lub nie znaleziono błędów. Prawidłowo zaprojektowany test, który przechodzi bez znalezienia błędu redukuje poziom ryzyka w systemie. Jeśli testowanie znajduje błędy i błędy

te zostają naprawione, jakość systemu informatycznego wzrasta. Samo testowanie nie podnosi jakości oprogramowania i dokumentacji.

Z poprzednich projektów powinny być wyciągane wnioski. Proces wytwórczy można doskonalić poprzez zrozumienie przyczyn błędów znalezionych w poprzednich projektach. W konsekwencji powinno zapobiec ponownemu pojawieniu się podobnych defektów a tym samym poprawić jakość przyszłych systemów.

Testowanie powinno być zintegrowane z innymi metodami zapewniania jakości (np. ze standardami kodowania, szkoleniami i analizą błędów).

### **1.1.5. Kiedy zakończyć testowanie (K2)**

Decyzja o tym, ile trzeba testować, powinna brać pod uwagę poziom ryzyka, włączając w to ryzyko techniczne, biznesowe i projektowe oraz ramy projektu takie jak czas i budżet. Temat ryzyka jest omówiony w rozdziale 5.

Testowanie powinno dostarczyć interesariuszom informacji wystarczającej do podjęcia decyzji o dopuszczeniu testowanego oprogramowania lub systemu do następnej fazy produkcji, przekazaniu go klientowi itp.

## 1.2. Co to jest testowanie (K2), 30 minut

### Terminologia

Kod, debugowanie, tworzenie (oprogramowania), wymaganie, przegląd, podstawa testów, przypadek testowy, testowanie, cele testu.

### Wprowadzenie

Zazwyczaj testowanie jest odbierane jako zajęcie polegające na wykonywaniu testów, czyli uruchamianiu oprogramowania. Jest to jedynie część, lecz nie całość, testowania.

Czynności związane z testowaniem, takie jak planowanie, nadzorowanie, ustalanie warunków testowych, projektowanie zadań testowych, porównywanie wyników, ocena kryteriów zakończenia, raporty dotyczące procesu testowego i testowanej aplikacji, zakończenie i zamykanie testów (po zakończeniu fazy testów), występują zarówno przed jak i po wykonaniu testów. Testowanie obejmuje również przeglądy dokumentów (w tym kodu) oraz analizę statyczną.

Zarówno testowanie statyczne jak i dynamiczne może być użyte do osiągnięcia podobnych celów. Oba podejścia dostarczą informacji koniecznych do poprawy testowanego oprogramowania oraz procesu tworzenia i testowania oprogramowania.

Można wyróżnić następujące cele testów:

Znajdowanie defektów

Budowanie zaufania odnośnie jakości oraz dostarczanie informacji o jakości

Zapobieganie awariom

Proces projektowania zadań testowych wykonywany wcześniej w cyklu wytwórczym (weryfikacja podstawy testów za pomocą projektowania testów) może zapobiec wprowadzaniu defektów do kodu. Przeglądy dokumentów (np. dokumentu wymagań) również pomagają zapobiegać pojawianiu się defektów.

Różne punkty widzenia podczas testowania mają na uwadze różne cele. Przykładowo, w testach prowadzonych przez programistów (np. testowanie modułowe, integracyjne i systemowe) głównym celem może być spowodowanie tylu awarii ile jest to możliwe, aby defekty w oprogramowaniu zostały zidentyfikowane i poprawione. W testach akceptacyjnych głównym celem może być potwierdzenie działania systemu zgodnie z oczekiwaniami, aby upewnić się, że system spełnia wymagania użytkowników. W niektórych przypadkach głównym celem testu może być ocena jakości oprogramowania (bez zamiaru naprawienia defektów), aby dostarczyć interesariuszom informacji na temat ryzyka związanego z wdrożeniem systemu w określonym czasie. Testy pielęgnacyjne często zawierają testy sprawdzające, czy nie wprowadzono żadnych nowych błędów podczas implementacji zmian. Podczas testów produkcyjnych głównym celem może być ocena wybranych charakterystyk takich jak niezawodność i dostępność.

Debugowanie i testowanie różnią się od siebie. Testowanie może ujawnić awarie spowodowane defektami. Debugowanie jest czynnością programistyczną wykonywaną w celu zidentyfikowania przyczyny defektu, poprawienia kodu i sprawdzenia czy defekt został poprawnie naprawiony. Późniejsze testowanie potwierdzające wykonywane przez testera zapewnia, że poprawka rzeczywiście usunęła awarię. Odpowiedzialność za każdą z tych czynności jest inna: testerzy testują, a programiści debugują.

Proces testowania i jego czynności są omówione w sekcji 1.4.

### 1.3. *Ogólne zasady testowania (K2), 35 minut*

#### Terminologia

Testowanie gruntowne

#### Zasady

Przez ponad 40 lat powstała pewna ilość zasad z zakresu testowania, które zawierają ogólne wytyczne cechujące każde testowanie.

##### Zasada 1 – Testowanie ujawnia błędy

Testowanie może pokazać, że istnieją defekty, lecz testując nie jesteśmy w stanie dowieść, że ich nie ma. Testowanie zmniejsza prawdopodobieństwo, że w oprogramowaniu pozostaną niezidentyfikowane defekty, ale nawet w przypadku, gdy żadne defekty nie zostaną znalezione, nie jest to dowodem na poprawność oprogramowania.

##### Zasada 2 – Testowanie gruntowne jest niemożliwe

Przetestowanie wszystkiego (wszystkich kombinacji wejść i warunków wstępnych) jest możliwe wyłącznie w odniesieniu do bardzo trywialnych przypadków. Określając zakres testów, zamiast skupiać się na testowaniu gruntownym, wykorzystujemy informacje o ryzyku i priorytetach.

##### Zasada 3 – Wczesne testowanie

Czynności testowe powinny rozpoczynać się tak wcześnie, jak to możliwe tylko w przypadku danego oprogramowania lub cyklu wytwarzania oprogramowania. Powinny być one również nastawione na osiągnięcie zdefiniowanych celów.

##### Zasada 4 – Kumulowanie się błędów

Większość defektów znalezionych podczas testowania przed wypuszczeniem oprogramowania lub powodujących awarie produkcyjne znajduje się w małej liczbie modułów.

##### Zasada 5 – Paradoks pestycydów

Jeżeli te same testy są ciągle powtarzane, ten sam zestaw przypadków testowych nie znajduje już żadnych nowych błędów. Żeby przezwyciężyć paradoks pestycydów, przypadki testowe muszą być regularnie przeglądane i korygowane. W celu sprawdzenia innych części oprogramowania lub systemu, aby potencjalnie znaleźć więcej błędów, powinny być dopisywane nowe, inne testy.

##### Zasada 6 – Testowanie jest zależne od kontekstu

Testowanie jest wykonywane w różny sposób w różnych sytuacjach. Na przykład, systemy krytyczne ze względu na bezpieczeństwo są testowane inaczej niż systemy typu *e-commerce*.

##### Zasada 7 – Mylne przekonanie o bezbłędności

Znalezienie i usunięcie błędów nie pomaga, jeżeli system jest nie nadający się do użytku i nie spełnia potrzeb oraz oczekiwań użytkownika.

## 1.4. Prosty proces testowy (K1), 35 minut

### Terminologia

Testowanie potwierdzające, warunki wyjścia, incydent, testowanie regresywne, podstawa testów, warunek testowy, pokrycie testowe, dane testowe, wykonanie testu, log testu, plan testów, strategia testowania, raport końcowy z testowania, testalia.

### Wprowadzenie

Najbardziej widoczną częścią testowania jest wykonanie testów. Jednak, żeby testy były efektywne i skuteczne, plany testów powinny uwzględniać również czas spędzony na planowaniu testów, projektowaniu przypadków testowych, przygotowaniu do wykonania testów oraz ocenie statusu wykonania testów.

Podstawowy proces testowy składa się z następujących głównych czynności:

- Planowanie i nadzór;
- Analiza i projektowanie;
- Implementacja i wykonanie;
- Ocena stopnia spełnienia warunków zakończenia i raportowanie;
- Czynności zamykające test.

Pomimo iż czynności te są logicznie sekwencyjne, w procesie mogą się zająć lub występować jednocześnie.

#### 1.4.1. Planowanie i nadzór nad testowaniem (K1)

Planowanie testów weryfikuje misję testowania, definiuje cele testowania oraz sposób ich osiągnięcia.

Nadzór nad testami to powtarzająca się czynność porównywania rzeczywistego postępu prac z planem i raportowanie wraz z informacją o odchyleniach od założeń. Nadzór nad testami pociąga za sobą czynności konieczne do wypełnienia misji i osiągnięcia celów projektu. Chcąc nadzorować testowanie, musimy je monitorować w sposób ciągły. Podczas planowania testów bierze się pod uwagę informacje zwrotne z monitorowania i nadzoru.

Planowanie testów zawiera następujące główne zadania:

- Ocena zakresu i ryzyka oraz identyfikacja celów testowania.
- Ocena metodyki testowania (techniki, testowane elementy, pokrycie, identyfikowanie i integracja zespołów zaangażowanych w testowanie, testalia).
- Ocena potrzebnych zasobów (np. ludzie, środowisko testowe, komputery).
- Wdrożenie polityki testów i/lub strategii testowania.
- Harmonogramowanie analizy testów i zadań projektowych.
- Harmonogramowanie implementacji, wykonania i oceny testów.
- Określenie kryteriów zakończenia.

Nadzór nad testami ma następujące główne zadania:

- Mierzenie i analiza rezultatów;
- Monitorowanie i dokumentowanie postępu prac, pokrycia testowego i kryteriów zakończenia;
- Rozpoczęcie prac naprawczych;

- Podejmowanie decyzji.

### **1.4.2. Analiza i projektowanie testów (K1)**

Analiza i projektowanie testów to czynności, w ramach których ogólne cele testowania zostają przekształcone w namacalne warunki testowe i projekty testów.

Analiza i projektowanie testów zawiera następujące główne zadania:

- Przeglądanie podstawy testów (takiej jak wymagania, architektura, projekt, interfejsy).
- Identyfikacja warunków testowych lub wymagań testowych oraz potrzebnych danych testowych na podstawie analizy przedmiotów testu, specyfikacji, zachowania i struktury.
- Projektowanie testów.
- Ocena testowalności wymagań i systemu.
- Projektowanie środowiska testowego i identyfikacja całej potrzebnej infrastruktury oraz narzędzi.

### **1.4.3. Implementacja i wykonanie testów (K1)**

Implementacja i wykonanie testów to czynności, w ramach których warunki testowe są przekształcane w przypadki testowe i testalia, oraz tworzone jest środowisko testowe.

Implementacja i wykonanie testów składają się z następujących czynności:

- Wytwarzanie i określanie priorytetów przypadków testowych, tworzenie danych testowych, pisanie procedur testowych oraz – opcjonalnie - przygotowywanie jarzma testowego i pisanie automatycznych skryptów testowych.
- Tworzenie scenariuszy testowych na podstawie przypadków testowych celem efektywnego wykonania testu.
- Weryfikacja poprawności utworzonego środowiska testowego.
- Wykonanie przypadków testowych ręcznie lub przy pomocy narzędzi, w zaplanowanej kolejności.
- Logowanie wyniku wykonania testu i rejestrowanie jednostek i wersji testowanego oprogramowania, narzędzi testowych oraz testaliów.
- Porównywanie wyników rzeczywistych z oczekiwanymi.
- Raportowanie rozbieżności jako zdarzeń i analizowanie ich celem znalezienia przyczyny wystąpienia (np. błąd w kodzie, w specyficznych danych testowych, w dokumencie testowym, lub pomyłka w sposobie wykonania testu).
- Powtarzanie czynności testowych jako rezultat akcji podjętych po stwierdzeniu rozbieżności. Na przykład: ponowne wykonanie testu, który poprzednio wykrył awarię celem potwierdzenia usunięcia usterki (testowanie potwierdzające); wykonanie poprawionego testu i/lub wykonanie testów celem upewnienia się, że nowe defekty nie zostały wprowadzone do niezmiennych obszarów oprogramowania lub, że usuwanie usterek nie powoduje innych awarii (testowanie regresywne).

### **1.4.4. Ocena kryteriów zakończenia oraz raportowanie testów (K1)**

Ocena stopnia spełnienia warunków zakończenia i raportowanie to czynności, w ramach których wykonanie testu jest oceniane pod względem zdefiniowanych celów. Tego typu czynności powinny być wykonywane dla każdego poziomu testów.

Ocena stopnia spełnienia warunków zakończenia i raportowanie składa się z następujących głównych czynności:

- Sprawdzenie dziennika testów w odniesieniu do warunków zakończenia określonych podczas planowania testów.

- Określenie, czy potrzebne jest więcej testów lub czy należy zmienić warunki zakończenia.
- Napisanie raportu końcowego z testowania dla interesariuszy.

#### **1.4.5. Czynności na zakończenie testowania (K1)**

W ramach czynności zamykających testowanie zbierane są dane z zakończonych czynności testowych celem gromadzenia doświadczenia, testaliów, faktów i liczb. Na przykład, kiedy oprogramowanie zostaje przekazane klientowi, projekt testowy jest zakończony (lub anulowany), krok milowy zostaje osiągnięty lub zamknięta zostaje nowa wersja oprogramowania po pielęgnacji.

Do czynności zamykających test zaliczamy:

- Sprawdzenie, które zaplanowane dostawy zostały dostarczone
- Zamknięcie raportów incydentów lub zgłoszenie zmian do tych, które pozostały otwarte
- Dokumentowanie akceptacji systemu.
- Zakończenie i archiwizacja testaliów, środowiska testowego i infrastruktury testowej celem ponownego wykorzystania.
- Przekazanie testaliów do zespołu serwisowego.
- Analiza wniosków na potrzeby przyszłych projektów, oraz poprawy zdolności / dojrzałości testów.

## 1.5. Psychologia testowania (K2), 35 minut

### Terminologia

Testowanie niezależne

### Wprowadzenie

Nastawienie podczas testowania i przeglądania jest inne niż podczas analizowania i wytwarzania. Mając odpowiednie nastawienie programiści są w stanie testować własny kod. Rozdzielenie odpowiedzialności między twórcami kodu a testerami wykonywane jest zwykle, w celu zwiększenia nacisku na testy i dostarczanie dodatkowych korzyści, takich jak niezależne spojrzenie wyszkolonego i profesjonalnego zespołu testowego. Testowanie niezależne może mieć miejsce na każdym poziomie testowania.

Pewien stopień niezależności (dzięki któremu unika się autorskiego spojrzenia na testowany produkt) pozwala zwykle na większą skuteczność w znajdowaniu defektów i awarii.

Niezależność nie zastępuje jednak znajomości programu; programiści mogą znaleźć wiele defektów we własnym kodzie. Zdefiniowano kilka poziomów niezależności:

- Testy projektowane przez osobę (osoby), które pisały testowany program (niski poziom niezależności).
- Testy projektowane przez inną osobę (osoby) (np. z zespołu wytwórczego).
- Testy projektowane przez osobę (osoby) z innej grupy organizacyjnej (np. niezależny zespół testowy).
- Testy projektowane przez osobę (osoby) z innej organizacji lub z innego przedsiębiorstwa (np. outsourcing lub certyfikacja wykonywana przez jednostkę zewnętrzną).

Ludzie oraz projekty kierują się określonymi celami. Ludzie mają skłonność do dopasowywania planów do celów określonych przez kierownictwo lub innych interesariuszy, na przykład, do znajdowania defektów, bądź potwierdzania, że oprogramowanie działa. Stąd ważne jest dokładne i jasne określenie celów testowania.

Identyfikacja awarii podczas testowania może być postrzegana jako krytyka skierowana pod adresem produktu lub jego autora. Powoduje to, że testowanie jest często postrzegane jako czynność destrukcyjna, nawet, jeżeli jest ona bardzo konstruktywna w świetle zarządzania ryzykiem projektu. Szukanie awarii w systemie wymaga ciekawości, profesjonalnego pesymizmu, krytycznego spojrzenia, przywiązywania wagi do szczegółów, dobrej komunikacji z programistami i doświadczenia, na którym można oprzeć zgadywanie błędów.

Jeżeli błędy, defekty lub awarie są komunikowane w konstruktywny sposób, można uniknąć spięć pomiędzy testerami, analitykami, projektantami i programistami. Odnosi się to zarówno do przeglądania, jak również do testowania.

Tester i kierownik testów potrzebują dobrych zdolności interpersonalnych, aby móc informować o faktycznym stanie defektów, postępu prac oraz o ryzyku w konstruktywny sposób. Dla autorów oprogramowania lub dokumentu, informacja o błędzie może okazać się pomocna w doskonaleniu umiejętności. Znalezienie podczas testowania i usunięte defekty pozwolą zaoszczędzić czas i środki w przyszłości, i zmniejszą ryzyko projektu.

Problemy komunikacyjne mogą wystąpić zwłaszcza wtedy, gdy testerzy są postrzegani jako osoby przekazujące informacje wyłącznie o awariach. Istnieje kilka sposobów by poprawić komunikację i relacje pomiędzy testerami i innymi pracownikami:

- Raczej współpracować niż walczyć – przypominać wszystkim o wspólnym celu, jakim jest lepsza jakość produktów.

- Informować o nieprawidłowościach produktu w neutralny sposób, zorientowany na fakty, bez krytykowania osoby, która stworzyła produkt, na przykład, pisać obiektywne, faktyczne i rzeczowe raporty zdarzeń i nieprawidłowości znalezionych podczas przeglądów.
- Spróbować zrozumieć, co czuje inna osoba i dlaczego reaguje w sposób, w jaki reaguje.
- Upewnić się, że inna osoba zrozumiała to, co zostało powiedziane.

#### Referencje:

1.1.5 Black, 2001, Kaner, 2002

1.2 Beizer, 1990, Black, 2001, Myers, 1979

1.3 Beizer, 1990, Hetzel, 1998, Myers, 1979

1.4 Hetzel, 1998

1.4.5 Black, 2001, Craig, 2002

1.5 Black, 2001, Hetzel, 1998

## 2. Testowanie w cyklu życia oprogramowania (K2), 135 minut

***Celem rozdziału „Testowanie w cyklu życia oprogramowania” jest nauczanie, w poszczególnych paragrafach:***

### 2.1 Modele wytwarzania oprogramowania (K2)

- Powiązań między wytwarzaniem, testowaniem i budowanymi produktami w cyklu wytwarzania oprogramowania; podać przykłady uwzględniające właściwości produktu, projektu oraz ich kontekst. (K2)
- Konieczności dostosowywania modeli wytwarzania oprogramowania do właściwości projektu i produktu. (K1)
- Przyczyn testowania na różnych poziomach, oraz charakterystyk dobrego testowania niezależnie od modelu cyklu życia oprogramowania.

### 2.2 Poziomy testów (K2)

- Porównywania różnych poziomów testów: głównych założeń testów, typowych przedmiotów celów testów (np. funkcjonalne lub strukturalne), kto wykonuje testy, rodzaje znajdowanych defektów i awarii. (K2)

### 2.3 Typy testów: cele testowania (K2)

- Czterech typów testów - funkcjonalne, niefunkcjonalne, strukturalne i wynikające ze zmian. (K2)
- O testowaniu funkcjonalnym i strukturalnym występującym na każdym poziomie testów. (K1)
- Rozpoznawania i opisywania testów niefunkcjonalnych bazujących na wymaganiach niefunkcjonalnych. (K2)
- Rozpoznawania i opisywania typów testów bazujących na strukturze lub architekturze systemu. (K2)
- Przeznaczenia testowania potwierdzającego i testowania regresywnego. (K2)

### 2.4 Testowanie w fazie utrzymania (K2)

- Testowania w fazie utrzymania (testowanie istniejącego systemu) i testowania nowej aplikacji, na zasadzie porównania, z uwzględnieniem celów, powodów i zakresu testów. (K2)
- Przyczyn testowania w fazie utrzymania (modyfikacja, migracja lub wycofanie systemu). (K1)
- Roli testów regresywnych i analizy wpływu w utrzymaniu oprogramowania. (K2)

## 2.1. Modele wytwarzania oprogramowania (K2), 20 minut

### Terminologia

Oprogramowanie z półki, przyrostowy model wytwarzania, poziom testów, walidacja, weryfikacja, model „V”.

### Wprowadzenie

Testowanie nie ma sensu w oderwaniu od czynności procesu wytwarzania oprogramowania, z którym jest powiązane. Różne modele wytwarzania oprogramowania wymagają odmiennej metodologii testowania.

#### 2.1.1. Model V (K2)

Pomimo, że istnieją rozmaite warianty modelu „V”, powszechnie wykorzystywane podejście stosuje cztery poziomy testów, odpowiadające czterem poziomom wytwarzania oprogramowania.

W opracowaniu tym wykorzystywane są następujące cztery poziomy:

- Testowanie modułowe (jednostkowe);
- Testowanie integracyjne;
- Testowanie systemowe;
- Testowanie akceptacyjne.

W rzeczywistości model „V” może mieć inne definicje poziomów wytwarzania i testowania oprogramowania, zależnie od projektu lub rodzaju produktu. Przykładowo, po testowaniu modułowym może występować testowanie integracyjne modułów, lub po testowaniu systemowym testowanie integracyjne systemów.

Powstające w procesie wytwarzania oprogramowania produkty (scenariusze biznesowe lub przypadki użycia, specyfikacje wymagań, dokumentacja projektowa oraz kod źródłowy) stanowią często podstawę testów na jednym lub kilku poziomach testów. Produkty te definiowane są m.in. w modelu CMMI (Capability Maturity Model Integration) oraz w normie IEEE/IEC 12207 (Software Life Cycle Process). W trakcie powstawania poszczególnych produktów procesu może być wykonywana ich weryfikacja i walidacja (oraz wczesne projektowanie testów).

#### 2.1.2. Iteracyjne modele wytwarzania (K2)

Wytwarzanie w modelu iteracyjnym polega na kilkukrotnym – w formie kilku przejść mniejszych procesów – określaniu wymagań, projektowaniu, programowaniu i testowaniu systemu. Przykładami modeli iteracyjnych są: prototypowanie, szybkie wytwarzanie oprogramowania (*Rapid Application Development*, RAD), *Rational Unified Process* (RUP) oraz metodyki zwinne (*agile development*). W trakcie trwania iteracji tworzony przyrost systemu może być testowany na kilku poziomach. Kolejne przyrosty systemu, dodawane do części wytworzonych wcześniej, stopniowo tworzą system, który również powinien być poddany testowaniu. W każdej kolejnej iteracji coraz ważniejsze stają się testy regresywne. W każdej iteracji można wykonywać zarówno weryfikację jak i walidację.

#### 2.1.3. Testowanie w cyklu życia (K2)

Każdy cykl życia systemu posiada kilka cech charakterystycznych dobrego testowania: Dla każdej czynności wytwarzania systemu istnieje odpowiadająca jej czynność testowa. Każdy poziom testów ma specyficzne dla siebie cele.

Analiza i projektowanie testów dla danego poziomu powinny rozpoczynać się już podczas odpowiadającej im fazy wytwarzania.

Testerzy powinni uczestniczyć w przeglądach wczesnych wersji dokumentacji tworzonej podczas wytwarzania.

Poziomy testów można łączyć ze sobą lub organizować na różne sposoby zależnie od specyfiki projektu lub architektury systemu. Na przykład podczas integracji w system gotowego oprogramowania z półki zamawiający może przeprowadzać testowanie integracyjne na poziomie systemu (np. testy integracji z infrastrukturą i pozostałymi systemami albo testy odbiorcze i wdrożeniowe) oraz testowanie akceptacyjne (testy funkcjonalne i/lub niefunkcjonalne, testy użytkowników końcowych i/lub testy zdolności operacyjnej).

## 2.2. Poziomy testów (K2), 60 minut

### Terminologia

Testowanie alfa, testowanie beta, testowanie modułowe (zwane też testowaniem jednostkowym, modułowym lub programu), testy akceptacyjne zgodności z umową, sterowniki, testy w środowisku produkcyjnym, wymagania funkcjonalne, integracja, testowanie integracyjne, wymagania нефункционалне, operacyjne testy akceptacyjne, testy akceptacyjne zgodności legislacyjnej, testowanie odporności, zaślepki, testowanie systemowe, wytwarzanie sterowane testowaniem (*test-driven development*), środowisko testowe, testowanie akceptacyjne przez użytkownika.

### Wprowadzenie

Dla każdego poziomu testów można określić następujące parametry: ogólne cele, produkty procesu wytwarzania, z których wywodzą się przypadki testowe (podstawa testów), przedmiot testowania (co będzie testowane), znajdowane typowe defekty i awarie, wymagania dotyczące narzędzi testowego oraz wsparcia narzędziowego, metodologii testowania oraz odpowiedzialności.

#### 2.2.1. Testowanie modułowe (K2)

Celem testów modułowych jest poszukiwanie błędów oraz weryfikowanie dających się przetestować elementów oprogramowania, np. modułów, programów, obiektów, klas itd. Testy modułowe wykonuje się zwykle osobno dla każdego testowanego obiektu, choć zależy to od rodzaju systemu oraz przyjętego cyklu wytwarzania oprogramowania. W tej sytuacji niezbędne może być posługiwanie się sterownikami testowymi, symulatorami oraz zaślepkami.

Celem testów modułowych może być zarówno weryfikacja funkcjonalności modułu jak i jego właściwości нефункционалне, na przykład wykorzystania zasobów (czy nie powoduje wycieków pamięci itp.). Ponadto wykonywane mogą być testy odporności modułu oraz testowanie strukturalne w celu osiągnięcia pokrycia rozgałęzień. Przypadki testowe projektuje się na podstawie specyfikacji modułów, specyfikacji projektu oprogramowania lub modelu danych.

Testowanie modułowe zwykle wykonuje się mając dostęp do kodu źródłowego, środowiska deweloperskiego, oprogramowania wspomagającego testy jednostkowe lub debagera. Uczestniczy w nim zazwyczaj programista będący twórcą testowanego modułu, natomiast defekty są naprawiane natychmiast po znalezieniu, bez formalnego zgłaszania incydentów.

Jedną z metodologii organizacji testów modułowych polega na przygotowaniu automatycznych przypadków testowych zanim jeszcze przystąpi się do kodowania. Metodologia ta nazywana jest „najpierw przygotuj testy” lub „wytwarzanie sterowane testowaniem”. Sposób ten jest wysoce iteracyjny i polega na cyklicznym budowaniu automatycznych przypadków testowych, a następnie konstruowaniu i integracji niewielkich fragmentów kodu oraz wykonywaniu testów modułowych aż do spełnienia ustalonych kryteriów.

#### 2.2.2. Testowanie integracyjne (K2)

Podczas testowania integracyjnego testuje się interfejsy między modułami, interakcję z innymi częściami systemu (system operacyjny, system plików, sprzęt) oraz interfejsy do innych systemów.

Testy integracyjne mogą być wykonywane na kilku poziomach, a ich przedmiotem mogą być części rozmaitej wielkości. Przykładowo, testy integracyjne modułów sprawdzają interakcje między modułami oprogramowania, wykonuje się je po zakończeniu testów modułowych. Natomiast testy integracyjne systemów sprawdzają interakcję między różnymi systemami i mogą być wykonywane po zakończeniu testowania systemowego. W tej sytuacji organizacja wytwarzająca oprogramowanie zarządza tylko jedną częścią interfejsu, co powoduje trudności w przypadku zmian. Procesy biznesowe, zaimplementowane jako przepływy, mogą angażować wiele systemów. Duże znaczenie mogą odgrywać zagadnienia zgodności między różnymi platformami.

Im większy jest zakres integracji, tym trudniejsze może być określenie, który moduł lub system jest przyczyną danej awarii, co powoduje zwiększone ryzyko.

Systematyczne strategie integracji mogą opierać się na architekturze systemu (np. scalanie wstępujące lub zstępujące), funkcjach systemu, kolejności przetwarzania transakcji, oraz innych właściwościach systemu lub modułu. Aby ograniczyć ryzyko spowodowane zbyt późnym ujawnianiem błędów, zaleca się raczej integrację przyrostową niż skokową („big bang”).

W ramach testów integracyjnych wykonuje się także niekiedy testowanie właściwości нефункциональных, np. wydajności.

Niezależnie od poziomu integracji, testowanie powinno dotyczyć wyłącznie samej integracji. Przykładowo, integrując ze sobą moduły A i B, powinno się testować komunikację między tymi modułami, a nie ich funkcjonalność. Stosuje się podejście zarówno strukturalne jak i funkcjonalne.

Najlepiej byłoby, aby testerzy rozumieli architekturę systemu oraz mieli wpływ na planowanie integracji. Planowanie testów integracyjnych przed przystąpieniem do wytwarzania systemu i jego modułów pozwala na przyjęcie takiej kolejności wytwarzania, która umożliwia najsprawniejsze testowanie.

### **2.2.3. Testowanie systemowe (K2)**

Testowanie systemowe dotyczy działania całego systemu lub produktu, zgodnie z zakresem projektu lub programu.

Dla testowania systemowego środowisko powinno odwzorowywać w miarę możliwości jak najwierniej środowisko produkcyjne, tak, aby zminimalizować ryzyko, że nie zostaną znalezione błędy zależne od jego specyfikacji.

Przypadki testowe wykorzystywane w testowaniu systemowym można projektować na podstawie oceny ryzyka, specyfikacji wymagań, procesu biznesowego, przypadków użycia lub innych opisów działania systemu (na wysokim poziomie), jak również interakcji systemu z systemem operacyjnym i jego zasobami.

Testowanie systemowe powinno dotyczyć zarówno wymagań funkcjonalnych jak i нефункциональных. Wymagania opisuje się w formie tekstowej lub w formie modeli. Potrzebna jest także umiejętność radzenia sobie z niepełnymi lub nieudokumentowanymi wymaganiami. Testowanie funkcjonalne należy rozpocząć od projektowania przypadków testowych przy pomocy najstosowniejszej dla danego systemu techniki czarnoskrzynkowej (w oparciu o specyfikację). Przykładowo, tablica decyzyjna może posłużyć do systematycznego opisu zależności czynników opisanych w regułach biznesowych. Można też zastosować techniki testowania strukturalnego (białoskrzynkowe), aby ocenić staranność testowania w odniesieniu do pokrycia wybranych elementów, np. struktury menu lub nawigacji między stronami WWW (zob. rozdz. 4.)

Testowanie systemowe wykonywane jest często przez niezależny zespół testowy.

## 2.2.4. Testowanie akceptacyjne (K2)

Odpowiedzialność za testowanie akceptacyjne należy często do klientów lub do użytkowników, choć mogą w nim mieć udział także inni interesariusze.

Celem testowania akceptacyjnego jest osiągnięcie zaufania do systemu, jego części lub określonych wymagań нефункциональных. Głównym celem testów akceptacyjnych nie jest znajdowanie defektów. Testy akceptacyjne mogą służyć do oceny gotowości systemu do wdrożenia lub użytkowania, choć niekoniecznie są ostatnią fazą testowania. Przykładowo, testy integracyjne zewnętrzne mogą być wykonywane po testach akceptacyjnych.

Testowanie akceptacyjne może pojawić się na więcej niż jednym poziomie, np.:

- Oprogramowanie z półki może być poddane testom akceptacyjnym w trakcie instalacji lub integracji,
- Testy akceptacyjne użyteczności modułu mogą mieć miejsce w trakcie testów modułowych,
- Testowanie akceptacyjne poprawy funkcjonalności może być wykonane przed testem systemowym.

Najczęściej spotyka się następujące formy testów akceptacyjnych:

### Testowanie akceptacyjne przez użytkownika

Typowy cel to weryfikacja dostosowania do użycia, dokonywana przez użytkowników biznesowych.

### Operacyjne testy akceptacyjne

Akceptacja systemu przez jego administratorów, dotycząca:

- Tworzenia kopii zapasowych i odtwarzania z nich systemu;
- Odtworzenia systemu po awarii;
- Zarządzania kontami użytkowników;
- Okresowych kontroli zagrożeń zabezpieczeń.

### Testy akceptacyjne zgodności z umową (odbiorcze) i testy zgodności legislacyjnej

Testy akceptacyjne zgodności z umową (testy odbiorcze) wykonuje się na podstawie określonych w kontrakcie kryteriów odbioru dla oprogramowania dostarczanego na zamówienie. Kryteria te powinny być określone w trakcie uzgadniania kontraktu. Testy akceptacyjne zgodności legislacyjnej wykonuje się na podstawie obowiązujących przepisów prawnych, branżowych lub bezpieczeństwa.

### Testy alfa oraz testy beta (testy w środowisku produkcyjnym)

Producenci oprogramowania sprzedawanego z półki zwykle chcą uzyskać wyniki oceny produktu przez istniejących lub potencjalnych klientów z właściwego segmentu rynku przed wypuszczeniem go do sprzedaży. Testowanie alfa wykonywane jest w siedzibie producenta oprogramowania. Testowanie beta (testy w środowisku produkcyjnym) wykonywane jest w siedzibach użytkowników. Zarówno testowanie alfa jak i testowanie beta wykonywane jest przez potencjalnych użytkowników, nie przez twórców produktu.

W odniesieniu do systemów testowanych przed i po dostarczeniu do operacyjnego środowiska użytkowników firmy stosują również inne określenia, takie jak testy akceptacji produkcyjnej lub testy akceptacji instalacji.

## 2.3. Typy i cele testów (K2), 40 minut

### Terminologia

Automatyzacja, testowanie czarnoskrzynkowe, pokrycie kodu, testowanie potwierdzające, testowanie funkcjonalne, testowanie zgodności operacyjnej, testowanie obciążeniowe, testowanie utrzymywalności, testowanie wydajnościowe, testowanie przenaszalności, testowanie regresywne, testowanie niezawodności, testowanie zabezpieczeń, testowanie w oparciu o specyfikację, testowanie przeciążające, testowanie strukturalne, scenariusz testowy, testowanie użyteczności, testowanie białoskrzynkowe.

### Wprowadzenie

Testowanie można podzielić według tego, jakie są powody lub przyczyny wykonywania danej grupy testów.

Cechą testów należących do jednej kategorii jest określony, wspólny cel: np. przetestowanie danej funkcjonalności, przetestowanie właściwości takiej jak niezawodność lub użyteczność, wykonanie testów struktury albo architektury systemu, lub testowanie związane ze zamianami (weryfikacja naprawy defektu – testowanie potwierdzające; poszukiwanie niezamierzonych zmian – testowanie regresywne).

W testach strukturalnych lub funkcjonalnych często stosowane są modele. Przykładowo, w testach funkcjonalnych może być wykorzystany model przepływu procesu, model automatu skończonego (przejsć stanów) albo specyfikacja przygotowana w języku naturalnym. Dla testów strukturalnych może to być model przepływu starowania lub model struktury menu.

### 2.3.1. Testowanie funkcji (testowanie funkcjonalne) (K2)

Funkcje systemu, podsystemu lub modułu opisuje się w specyfikacji wymagań w formie przypadków użycia lub w formie specyfikacji funkcjonalnej. Bywają również nieudokumentowane. Funkcje te określają CO robi system.

Testy funkcjonalne dotyczą funkcji lub cech systemu (udokumentowanych lub znanych testerom) i wykonuje się je na wszystkich poziomach testów (np. testy modułu na podstawie specyfikacji modułu).

Techniki testowania w oparciu o specyfikację polegają na tym, że warunki i przypadki testowe projektuje się na podstawie specyfikacji funkcjonalnej oprogramowania lub systemu (zob. rozdz. 4). Testowanie funkcjonalne dotyczy zewnętrznego działania oprogramowania (testowanie czarnoskrzynkowe).

Jeden z typów testów funkcjonalnych - testowanie zabezpieczeń - weryfikuje funkcjonalność (np. zapory – „firewall”) służącą zabezpieczeniu przed zagrożeniami takimi jak wirusy czy nieuprawniony dostęp osób z zewnątrz.

### 2.3.2. Testowanie właściwości (testowanie niefunkcjonalne) (K2)

W skład testów niefunkcjonalnych wchodzi między innymi: testowanie wydajności, testowanie obciążeniowe, testowanie przeciążające, testowanie użyteczności, testowanie współdziałania, testowanie utrzymywalności, testowanie niezawodności oraz testowanie przenaszalności. Testowanie to określa JAK system działa.

Testowanie niefunkcjonalne można przeprowadzać na wszystkich poziomach testów. Określenie testowanie niefunkcjonalne odnosi się do testów niezbędnych do pomiaru charakterystyk systemu i oprogramowania, które dają się skwantyfikować na skali (np. czas

odpowiedzi w testowaniu wydajnościowym). Testowanie tego typu odwołuje się do modelu jakości takiego jak np. ISO 9126 „Software Engineering – Software Product Quality”.

### **2.3.3. Testowanie struktury/architektury systemu (testowanie strukturalne) (K2)**

Testy strukturalne (białoskrzynkowe) wykonuje się na wszystkich poziomach testów. Techniki testowania strukturalnego najlepiej jest stosować po technikach w oparciu o specyfikację, aby zmierzyć staranność testowania poprzez ocenę pokrycia wybranego rodzaju struktur.

Pokrycie testowe to miara określająca stopień wykonania danej struktury przez scenariusz testowy, obliczona jako odsetek pokrytych elementów. Jeśli pokrycie nie wynosi 100%, aby przetestować wcześniej pominięte elementy, można zaprojektować dodatkowe przypadki testowe. Techniki pomiaru pokrycia omówione są w Rozdziale 4.

Na wszystkich poziomach testów, a zwłaszcza w testach modułowych oraz testach integracyjnych modułów, stosuje się narzędzia do mierzenia pokrycia takich elementów kodu, jak instrukcje lub decyzje. Testowanie strukturalne można wykonywać zgodnie z architekturą systemu, np. hierarchią wywołań.

Podejście strukturalne może być również stosowane na poziomie testów systemowych, testów integracji systemów lub testów akceptacyjnych, np. w odniesieniu do modelu biznesowego albo struktury menu.

### **2.3.4. Testowanie związane ze zmianami (testowanie potwierdzające i regresywne) (K2)**

Po wykryciu i naprawieniu defektu, aby zweryfikować (potwierdzić), że usunięcie defektu zakończyło się powodzeniem, oprogramowanie należy przetestować ponownie. Nazywa się to testowaniem potwierdzającym. Debugowanie (naprawa defektu) należy do wytwarzania, nie do testowania.

Testowanie regresywne polega na ponownym testowaniu już przetestowanego programu po jego modyfikacji po to, aby ujawnić możliwe defekty powstałe – lub odkryte – w wyniku wprowadzonej zmiany lub zmian. Defekty te mogą występować albo w testowanym oprogramowaniu, albo w innym – mającym lub nie, związek z przedmiotem testów – module oprogramowania. Testowanie regresywne wykonuje się po zmianie oprogramowania lub jego środowiska operacyjnego. Zakres testów regresywnych zależy od tego, jak duże jest ryzyko, że znajdzie się usterki w oprogramowaniu, które wcześniej działało poprawnie.

Testy, które mają być stosowane do testowania potwierdzającego i testowania regresywnego muszą być powtarzalne.

Testowanie regresywne można przeprowadzać na wszystkich poziomach testów. W skład testów regresywnych wchodzi wszelkie typy testów: funkcjonalne, właściwości i strukturalne. Ponieważ zestawy testów do testów regresywnych wykorzystuje się wielokrotnie i są one stosunkowo niezmiennie, są dobrymi kandydatami do automatyzacji.

## **2.4. Testowanie w fazie utrzymania (K2), 15 minut**

### Terminologia

Analiza wpływu, testowanie w fazie utrzymania, migracja, modyfikacje, wycofanie systemu.

### Wprowadzenie

Po wdrożeniu wiele systemów informatycznych jest wykorzystywanych przez lata, a nawet dziesiątki lat. W tym czasie system i jego środowisko poddawane są licznym naprawom, zmianom i rozbudowie. Testowanie w fazie utrzymania wykonywane jest na systemie produkcyjnym, a jego powodem są modyfikacje, migracje lub wycofanie oprogramowania czy systemu.

Modyfikacje przeprowadzane podczas utrzymania systemu mają różne przyczyny: rozbudowa i udoskonalanie systemu (np. w formie planowanych wersji), modyfikacje naprawcze i awaryjne, a także różnorodne zmiany środowiska (kolejne planowane wersje systemu operacyjnego lub bazy danych lub łaty usuwające nowoodkryte usterki zabezpieczeń systemu operacyjnego).

Testowanie w fazie utrzymania wykonywane w trakcie migracji (np. na inną platformę) powinno dotyczyć zarówno testów zdolności operacyjnej dla nowego środowiska, jak i testów zmodyfikowanego oprogramowania.

Testowanie w fazie utrzymania wykonywane w związku z wycofaniem systemu może obejmować testy migracji danych lub – jeśli dane muszą być przechowywane przez długi czas – testy ich archiwizacji.

Oprócz testowania wykonanych zmian, testy w fazie utrzymania obejmują także pełne testy regresywne niezmiennych części systemu. Zakres testów regresywnych zależy od ryzyka zmiany, wielkości systemu oraz zakresu zmiany. Zależnie od rodzaju zmian, testowanie w fazie utrzymania wykonywane jest na którymkolwiek – lub na wszystkich – poziomach testów oraz dotyczy któregośkolwiek – lub wszystkich – typów testów.

Analiza wpływu polega na określeniu, w jaki sposób zmiany mogą wpłynąć na system. Wynik jej określa, jak obszerne testy regresywne należy wykonać.

Przyczyną trudności w testach w fazie utrzymania bywa przestarzała czy wręcz zaginiona specyfikacja systemu.

### Referencje

2.1.3 CMMI, Craig, 2002, Hetzel, 1998, IEEE 12207

2.2 Hetzel, 1998

2.2.4 Copeland, 2004, Myers, 1979

2.3.1 Beizer, 1990, Black, 2001, Copeland, 2004

2.3.2 Black, 2001, ISO 9126

2.3.3 Beizer, 1990, Copeland, 2004, Hetzel, 1998

2.3.4 Hetzel, 1998, IEEE 829

2.4 Black, 2001, Craig, 2002, Hetzel, 1998, IEEE 829

### **3. Statyczne techniki testowania (K2), 60 minut**

***Celem rozdziału „Statyczne techniki testowania” jest nauczenie, w poszczególnych paragrafach:***

#### **3.1 Przeglądy i proces testowy (K2)**

- Rozpoznawania produktów oprogramowania badanych przez różne statyczne techniki testowania. (K1)
- Znaczenia i wartości rozważanych statycznych technik testowania w celu oceny produktów oprogramowania. (K2)
- Różnic pomiędzy statycznymi i dynamicznymi technikami testowania. (K2)

#### **3.2 Proces przeglądu (K2)**

- Faz, ról i odpowiedzialności typowego formalnego przeglądu. (K1)
- Różnic pomiędzy rodzajami przeglądu: przegląd nieformalny, przegląd techniczny, przejrzanie i inspekcja. (K2)
- Czynników udanego wykonania przeglądu. (K2)

#### **3.3 Analiza statyczna wsparta narzędziowo (K2)**

- Celu analizy statycznej i porównania jej z testowaniem dynamicznym. (K2)
- Typowych defektów i pomyłek wykrytych dzięki analizie statycznej i porównania ich z przeglądami i testowaniem dynamicznym. (K1)
- Typowych zalet analizy statycznej. (K1)
- Typowych błędów kodu i projektu, które można zidentyfikować za pomocą narzędzi do analizy statycznej. (K1)

### 3.1. Przeglądy i proces testowy(K2), 15 minut

#### Terminologia

Testowanie dynamiczne, przeglądy, analiza statyczna.

#### Wprowadzenie

Statyczne techniki testowania nie wykonują testowanego programu; są to techniki ręczne (przeglądy) lub zautomatyzowane (analiza statyczna).

Przeglądy są sposobem testowania produktów pracy programowej (wliczając w to kod) i mogą być wykonane przed wykonaniem testu dynamicznego. Defekty wykryte podczas przeglądów na wczesnym etapie cyklu życia oprogramowania są często dużo tańsze do usunięcia niż defekty wykryte podczas wykonywania późniejszych testów (np. defekty znalezione w wymaganiach).

Przeгляд mógłby być wykonany w całości jako czynność ręczna, jednakże istnieje wsparcie narzędziowe dla tego procesu. Główna ręczna czynność to zbadać produkt i zebrać uwagi na jego temat. Można przejrzeć każdy produkt pracy programowej, wliczając w to specyfikacje wymagań, specyfikacje projektu, kod, plany testów, specyfikacje testów, przypadki testowe, skrypty testowe, instrukcje użytkownika czy strony webowe.

Do zalet przeglądów wlicza się: wczesne wykrycie i naprawę defektu, poprawę produktywności oprogramowania, skrócony czas tworzenia oprogramowania, skrócony czas i koszt testowania, zmniejszenie defektów oraz lepszą komunikację. Podczas wykonywania przeglądów można wykryć opuszczenia (np. w wymaganiach), co jest dużo mniej prawdopodobne podczas testowania dynamicznego.

Przeglądy, analiza statyczna i testowanie dynamiczne mają ten sam cel – zidentyfikowanie defektów. Są to czynności uzupełniające się: różne techniki mogą znaleźć różne rodzaje defektów efektywnie i wydajnie. W przeciwieństwie do testowania dynamicznego, przeglądy znajdują raczej defekty niż awarie.

Typowe defekty łatwiejsze do znalezienia w przeglądach niż w testowaniu dynamicznym to: odchylenia od standardów, błędy wymagań, błędy projektu, niewystarczająca zdolność do konserwacji, nieprawidłowe specyfikacje interfejsów.

## 3.2. Proces przeglądu (K2), 25 minut

### Terminologia

Kryteria wejściowe, warunki wyjścia, przegląd formalny, przegląd nieformalny, inspekcja, rozpoczęcie, miary, moderator/prowadzący inspekcję, przegląd koleżeński, przeglądający, proces przeglądu, protokolant, przegląd techniczny, przejrzanie.

### Wprowadzenie

Przeglądy mogą być od bardzo nieformalnych do bardzo formalnych (dobrze zorganizowanych i uregulowanych). Formalność procesu przeglądu związana jest z takimi czynnikami jak dojrzałość procesu programowania, prawne lub ustalone wymagania oraz potrzeby audytu.

Sposób, w jaki przegląd jest przeprowadzany, zależy od uzgodnionego celu przeglądu (np. znaleźć defekt, zrozumieć lub wywołać dyskusję i osiągnąć decyzję poprzez konsensus).

### 3.2.1. Fazy formalnego przeglądu (K1)

Typowy formalny przegląd ma następujące główne fazy:

- Planowanie: wybór osób, przypisanie ról; określenie kryteriów wejściowych i warunków wyjścia (dla bardziej formalnych typów przeglądu np. inspekcji); wybór części dokumentów do przejrzania.
- Rozpoczęcie: dystrybucja dokumentów; wyjaśnienie uczestnikom celów, procesu i dokumentów; sprawdzenie kryteriów wejściowych (dla bardziej formalnych typów przeglądu).
- Indywidualne przygotowanie: praca wykonana samodzielnie przez każdego z uczestników przed spotkaniem przeglądownym, zanotowanie potencjalnych błędów, pytań i komentarzy.
- Spotkanie przeglądowne: dyskusja lub zapis z udokumentowanymi wynikami lub protokołami (dla bardziej formalnych typów przeglądu). Uczestnicy spotkania mogą po prostu zanotować defekty, przedstawić zalecenia odnośnie obsługi defektów lub poczynić decyzje w sprawie defektów.
- Obróbka: ustalenie znalezionych defektów - zazwyczaj wykonywane przez autora.
- Dalsza część: sprawdzenie, czy defekty zostały zaadresowane; zebranie metryk i sprawdzenie warunków wyjścia (dla bardziej formalnych typów przeglądów).

### 3.2.2. Role i odpowiedzialności (K1)

Typowy formalny przegląd może zawierać następujące role:

- Menedżer: decyduje o wykonaniu przeglądów, przydziela czas w harmonogramach projektów i określa, czy zostały uwzględnione cele przeglądu.
- Moderator: osoba, która prowadzi przegląd dokumentu lub zestawu dokumentów, wliczając w to planowanie przeglądu, przebieg spotkania i dalszą część po spotkaniu. Jeżeli konieczne, moderator może prowadzić mediacje pomiędzy różnymi punktami widzenia i często jest osobą, na której spoczywa sukces przeglądu.
- Autor: autor lub osoba odpowiedzialna za przeglądany dokument lub dokumenty.
- Przeglądający: osoby ze specyficznym technicznym lub biznesowym przygotowaniem (zwane także kontrolerami lub inspektorami), które po niezbędnym przygotowaniu, identyfikują i opisują wyniki badań (np. błędy) przeglądanej produktu. Przeglądający powinni być wybrani w taki sposób, aby reprezentować różne perspektywy i role. Uczestniczą we wszelkich spotkaniach przeglądowych.

- Protokolant (lub rejestrator): dokumentuje wszystkie kwestie, problemy i otwarte punkty, które zidentyfikowano podczas spotkania.

Spojrzenie na dokumenty z różnych perspektyw oraz używanie list kontrolnych może uczynić przeglądy bardziej efektywnymi i wydajnymi. Przykładowo, lista kontrolna oparta na perspektywach takich jak użytkownik, osoba zajmująca się utrzymaniem oprogramowania, tester lub wdrożeniowiec albo lista kontrolna typowych problemów wymagań.

### 3.2.3. Rodzaje przeglądów (K2)

Pojedynczy dokument może być przedmiotem więcej niż jednego przeglądu. Jeżeli wykonuje się więcej niż jeden rodzaj przeglądu to ich kolejność może być różna. Na przykład, przegląd nieformalny można przeprowadzić przed przeglądem technicznym, inspekcję odnośnie specyfikacji wymagań można przeprowadzić przed przejrzeniem ich z klientami. Główne cechy, opcje i cele powszechnych rodzajów przeglądu to:

#### Przegląd nieformalny

Kluczowe właściwości:

- Brak formalnego procesu;
- Możliwość zastosowania przy programowaniu parami lub do przeglądów projektu i kodu przez kierownika zespołu;
- Opcjonalnie: może być udokumentowany;
- Może różnić się w przydatności w zależności od przeglądanego;
- Główny cel: niedrogi sposób uzyskania pewnej korzyści.

#### Przejrzenie

Kluczowe właściwości:

- Spotkanie prowadzone przez autora;
- Scenariusze, przeglądy kodu, grupa koleżeńska;
- Otwarte dyskusje;
- Opcjonalnie: przygotowanie przeglądających przed spotkaniem, raport z przeglądu, lista wykrytych rzeczy i protokolant, (który nie jest autorem);
- W praktyce może być zróżnicowany od zupełnie nieformalnego do bardzo formalnego;
- Główne cele: nauka, zrozumienie, znalezienie defektu.

#### Przegląd techniczny

Kluczowe właściwości:

- Udokumentowany, nastawiony na wykrycie błędów proces, w którym biorą udział koledzy i eksperci techniczni;
- Może być przeprowadzony jako przegląd koleżeński bez uczestnictwa zarządu;
- Idealnie prowadzony przez wyszkolonego moderatora (nie autora);
- Wstępne przygotowanie przed spotkaniem;
- Opcjonalnie: użycie list kontrolnych, raportu przeglądu, list wykrytych rzeczy oraz uczestnictwo zarządu;
- W praktyce może być zróżnicowany od zupełnie nieformalnego do bardzo formalnego;
- Główne cele: przedyskutować, podjąć decyzje, ocenić alternatywy, znaleźć defekty, rozwiązać problemy techniczne oraz sprawdzić zgodność ze specyfikacjami i standardami.

#### Inspekcja

Kluczowe właściwości:

- Prowadzona przez wyszkolonego moderatora (nie autora);
- Zwykle wykonywana przez osoby o podobnych do autora kompetencjach;
- Określone role;
- Zawiera miary;
- Formalny proces oparty na regułach oraz listach kontrolnych z kryteriami wejściowymi i warunkami wyjścia;
- Wstępne przygotowanie przed spotkaniem;
- Raport z inspekcji, lista wykrytych rzeczy;
- Formalny proces kontroli po wykonaniu napraw;
- Opcjonalnie: ulepszenie procesu oraz czytelnik;
- Główny cel: znaleźć defekty.

### **3.2.4. Czynniki powodzenia przeglądów (K2)**

O powodzeniu przeglądów stanowią następujące czynniki:

- Każdy przegląd ma na początku jasno określony cel.
- Do celów przeglądu angażuje się właściwe osoby.
- Znalezione błędy przyjmowane są z radością i mówi się o nich obiektywnie.
- Ma się do czynienia z wpływami ludzkimi i aspektami psychologicznymi (np. jest to pozytywnym doświadczeniem dla autora).
- Stosuje się techniki przeglądowe, które są odpowiednie dla typu i poziomu produktów pracy programowej oraz osób przeglądających.
- Jeśli stosownym jest zwiększyć efektywność identyfikacji defektu korzysta się z list kontrolnych i ról.
- Stosuje się szkolenia w technikach przeglądowych, szczególnie w bardziej formalnych technikach takich jak inspekcja.
- Zarząd popiera dobry proces przeglądu (np. poprzez uwzględnienie odpowiedniego czasu dla czynności przeglądu w harmonogramach projektu).
- Kładzie się nacisk na naukę i ulepszenie procesu przeglądu.

### 3.3. Analiza statyczna przy pomocy narzędzi (K2), 20minut

#### Terminologia

Kompilator, złożoność, przepływ sterowania, przepływ danych, analiza statyczna.

#### Wprowadzenie

Celem analizy statycznej jest wykryć defekt w kodzie źródłowym oprogramowania lub w modelach oprogramowania. Analiza statyczna przeprowadzana jest bez równoczesnego wykonywania badanego oprogramowania przez zewnętrzne narzędzie (testowanie dynamiczne wykonuje kod oprogramowania). Analiza statyczna może zlokalizować defekty trudne do wykrycia w innym testowaniu. Tak jak przeglądy, analiza statyczna wykrywa raczej defekty niż awarie. Narzędzia analizy statycznej analizują kod programu (np. przepływ sterowania i przepływ danych) na tyle skutecznie na ile dobrze generowane jest wyjście takie jak HTML i XML.

Zalety analizy statycznej to:

- Wczesne wykrycie błędów przed wykonaniem testu.
- Dzięki obliczeniu miar, takich jak wysoka wartość miary złożoności, wczesne ostrzeżenie o podejrzanych aspektach kodu lub projektu.
- Identyfikacja defektów, których nie daje się łatwo wykryć podczas testowania dynamicznego.
- Wykrycie zależności i niespójności w modelach oprogramowania takich jak linki.
- Lepsza zdolność do pielęgnacji kodu i projektu.
- Zapobieganie błędom, jeżeli zdobyto wiedzę podczas tworzenia oprogramowania.

Do typowych błędów wykrytych przy pomocy narzędzi analizy statycznej należą:

- Odwołanie się do zmiennej z nieokreśloną wartością;
- Niespójny interfejs pomiędzy modułami;
- Zmienne, których nigdy nie użyto;
- Nieosiągalny (martwy) kod;
- Naruszenie standardów programowania;
- Słabe punkty bezpieczeństwa;
- Błędy składni kodu i modeli oprogramowania.

Narzędzia do analizy statycznej używane są zwykle przez programistów (sprawdzanie względem wcześniej określonych zasad i standardów programowania) przed i podczas testowania modułowego i integracyjnego oraz przez projektantów podczas modelowania oprogramowania. Narzędzia do analizy statycznej mogą wytworzyć dużą liczbę komunikatów ostrzegawczych. Wymagają one dobrego zarządzania, aby użycie narzędzia było najbardziej efektywne.

Kompilatory również wykonują elementy analizy statycznej, w tym obliczanie miar dotyczących kodu źródłowego.

#### Referencje

3.2 IEEE 1028

3.2.2 Gilb, 1993, van Veenendaal, 2004

3.2.4 Gilb, 1993, IEEE 1028

3.3 Van Veenendaal, 2004

## 4. Techniki projektowania testów (K3), 255 minut

**Celem rozdziału „Techniki projektowania testów” jest nauczenie, w poszczególnych paragrafach:**

### 4.1 Identyfikacja warunków testowych i projektowanie przypadków testowych (K3)

- Różnic pomiędzy specyfikacją projektowania testów, specyfikacją przypadków testowych i specyfikacją procedury testowej. (K1)
- Pojęć: warunek testowy, przypadek testowy i procedura testowa. (K2)
- Pisania przypadków testowych: (K3)
  - Pokazujących proste śledzenie powiązań z wymaganiami;
  - Zawierających oczekiwany wynik
- Tłumaczenia przypadków testowych na poprawnie skonstruowaną specyfikację procedury testowej na poziomie szczegółowości odpowiednim dla wiedzy testerów. (K3)
- Pisania harmonogramu wykonania testu dla danego zestawu przypadków testowych, uwzględniając priorytety oraz zależności techniczne i logiczne. (K3)

### 4.2 Kategorie technik projektowania testów (K2)

- Powodów, dla których projektowanie przypadków testowych w oparciu o specyfikację (podejście czarnoskrzynkowe) i projektowanie przypadków testowych w oparciu o strukturę (podejście białoskrzynkowe) jest użyteczne oraz pokazania podstawowych technik dla każdego z nich. (K1)
- Charakterystyk i różnic pomiędzy: testowaniem w oparciu o specyfikację, testowaniem w oparciu o strukturę i testowaniem w oparciu o doświadczenia. (K2)

### 4.3 Techniki testowania w oparciu o specyfikację czyli czarnoskrzynkowe (K3)

- Pisania przypadków testowych dla zadanych modeli oprogramowania wykorzystując następujące techniki projektowania testów: (K3)
  - Podział na klasy równoważności;
  - Analiza wartości brzegowych;
  - Testowanie w oparciu o tablice decyzyjne;
  - Diagramy przejść pomiędzy stanami.
- Głównych przyczyn zastosowania każdej z czterech technik, jakiego poziomu i typu testy mogą wykorzystywać tę technikę i jak można zmierzyć pokrycie. (K2)
- Koncepcji i zalet testowania w oparciu o przypadki użycia. (K2)

### 4.4 Techniki testowania w oparciu o strukturę lub białoskrzynkowe (K3)

- Koncepcji i znaczenia pokrycia kodu. (K2)
- Koncepcji pokrycia instrukcji kodu i pokrycia decyzji oraz, że koncepcje takie można wykorzystać na innych poziomach testów niż testowanie modułowe (np. na poziomie systemu w procesach biznesowych). (K2)
- Napisania przypadków testowych na podstawie podanych przepływów sterowania wykorzystując następujące techniki projektowania testów: (K3)
  - Testowanie instrukcji;
  - Testowanie decyzyjne.
- Oszacowania pokrycia instrukcji kodu i decyzji po zakończeniu testów. (K3)

#### 4.5 Techniki w oparciu o doświadczenie (K2)

- Powodów napisania przypadków testowych w oparciu o intuicję, doświadczenie i wiedzę o podstawowych defektach. (K1)
- Porównywania technik testowania opartych na doświadczeniu z technikami w oparciu o specyfikację. (K2)

#### 4.6 Wybór technik testowych (K2)

- Czynników, które wpływają na wybór właściwej techniki projektowania testów dla szczególnego rodzaju problemu, takiego jak typ systemu, ryzyko, wymagania klienta, wzorzec modelowania przypadków użycia, modele wymagań lub wiedza testera. (K2)

#### **4.1. Identyfikowanie warunków testowych i projektowanie przypadków testowych (K3), 15 minut**

##### Terminologia

Przypadki testowe, specyfikacja przypadku testowego, warunek testowy, dane testowe, specyfikacja procedury testowej, skrypt testowy, śledzenie powiązań.

##### Wprowadzenie

Proces identyfikowania warunków testowych i projektowania testów składa się z kilku kroków:

- Projektowanie testów przez identyfikowanie warunków testowych.
- Specyfikowanie przypadków testowych.
- Specyfikowanie procedur testowych.

Proces można wykonać w różnoraki sposób, od bardzo nieformalnego z małą ilością dokumentacji lub bez niej do bardzo formalnego (jaki został opisany w tej sekcji). Poziom sformalizowania zależy od kontekstu testowania, wliczając w to organizację, dojrzałość procesów testowania i programowania, ograniczenia czasowe oraz zaangażowane osoby. Podczas projektowania testów analizuje się podstawę testów, w celu określenia, co testować i w celu identyfikacji warunków testowych. Warunek testowy jest definiowany jako element lub zdarzenie, które powinno być zweryfikowane przez jeden lub więcej przypadków testowych (np. funkcja, transakcja, cecha lub element konstrukcyjny).

Ustawienie śledzenia powiązań pomiędzy warunkami testowymi a specyfikacją lub wymaganiami umożliwia analizę wpływu w momencie zmiany wymagań oraz analizę pokrycia wymagań określonym zestawem testów. Podczas projektowania testów szczegółowe podejście testowe jest budowane na podstawie zidentyfikowanego ryzyka (patrz rozdział 5, gdzie więcej o analizie ryzyka).

W czasie specyfikacji przypadków testowych dane i przypadki testowe są szczegółowo budowane i opisywane z wykorzystaniem technik projektowania testów. Przypadek testowy składa się z zestawu wartości wejściowych, warunków początkowych, oczekiwanych wyników i warunków końcowych, zaprojektowanych w celu pokrycia pewnych warunków testowych. „Standard Dokumentacji Testowej Oprogramowania” („Standard for Software Test Documentation” IEEE 829) opisuje zawartość specyfikacji projektowania testów oraz zawartość specyfikacji przypadków testowych.

Oczekiwane wyniki powinny powstawać jako część specyfikacji przypadku testowego i powinny zawierać wartości wyjścia, zmiany danych i stanów oraz inne wyniki wykonania testu. Jeżeli oczekiwane wyniki nie będą określone wówczas uzyskany wynik, chociaż błędny, może być zinterpretowany jako prawidłowy. W idealnej sytuacji oczekiwane wyniki powinny być zawsze określone przed wykonaniem testu.

Przygotowane przypadki testowe układa się w kolejności wykonywania; jest to specyfikacja procedury testowej. Procedura ta (lub ręczny skrypt testowy) określa kolejność działań przy wykonaniu testu. Jeżeli testy przeprowadzane są z wykorzystaniem narzędzia wspierającego wykonanie testu, kolejność działań określana jest w skrypcie testowym (który jest zautomatyzowaną procedurą testową).

Różne procedury testowe i zautomatyzowane skrypty testowe są następnie wstawiane w harmonogram wykonania testu, który określa kolejność ich wykonania, oraz kiedy i przez kogo mają być one wykonane. W harmonogramie wykonania testu bierze się pod uwagę takie czynniki jak testy regresyjne, priorytety, zależności techniczne i logiczne.

## 4.2. Kategorie technik projektowania testów (K2), 15 minut

### Terminologia

Techniki czarnoskrzynkowe, techniki oparte na doświadczeniu, techniki oparte na specyfikacji, techniki oparte na strukturze, techniki białoskrzynkowe.

### Wprowadzenie

Celem techniki projektowania testów jest zidentyfikowanie warunków testowych i przypadków testowych.

Klasycznym podziałem technik testowych jest podział na technikę czarnoskrzynkową i technikę białoskrzynkową. Techniki czarnoskrzynkowe (zwane również technikami opartymi na specyfikacji) znajdują i wybierają warunki i przypadki testowe na podstawie analizy dokumentacji (podstawy testów) zarówno funkcjonalnej jak i niefunkcjonalnej bez odwoływania się do jego wewnętrznej struktury. Techniki białoskrzynkowe (zwane również technikami strukturalnymi lub opartymi na strukturze) oparte są na analizie wewnętrznej struktury modułu lub systemu.

Pewne techniki mieszczą się wyraźnie w jednej kategorii; inne mają elementy z więcej niż jednej kategorii. Konspekt ten odnosi się do podejść opartych na specyfikacji i opartych na doświadczeniu jako technik czarnoskrzynkowych oraz do podejścia opartego na strukturze jako techniki białoskrzynkowej.

Podstawowe cechy technik opartych na specyfikacji:

- Modele formalne lub nieformalne są wykorzystywane do specyfikowania rozwiązywanych problemów, systemu lub jego modułów.
- Przypadki testowe mogą być wytwarzane systematycznie na podstawie modeli.

Podstawowe cechy technik opartych na strukturze:

- Przypadki testowe pochodzą z informacji o tym jak skonstruowane jest oprogramowanie, na przykład z kodu lub projektu.
- Dla istniejących przypadków testowych można mierzyć stopień pokrycia oprogramowania i można systematycznie tworzyć kolejne przypadki testowe w celu poprawienia pokrycia.

Podstawowe cechy technik opartych na doświadczeniu:

- Do tworzenia przypadków testowych wykorzystuje się wiedzę i doświadczenie osób.
- Wiedza testerów, programistów, użytkowników i innych interesariuszy na temat oprogramowania, jego wykorzystywania i środowiska;
- Wiedza o podobnych defektach i ich dystrybucji.

### **4.3. Techniki na podstawie specyfikacji lub czarnoskrzynkowe (K3), 120 minut**

#### **Terminologia**

Analiza wartości brzegowych, testowanie w oparciu o tablicę decyzyjną, podział na klasy równoważności, testowanie przejść pomiędzy stanami, testowanie w oparciu o przypadki użycia.

#### **4.3.1. Podział na klasy równoważności (K3)**

Elementy wejściowe do oprogramowania lub systemu dzieli się na grupy o podobnym charakterze - prawdopodobnie będą one przetwarzane przez system w ten sam sposób. Klasy równoważności można znaleźć zarówno dla danych poprawnych jak i niepoprawnych, tzn. wartości, które powinny być odrzucone. Klasy równoważności można zidentyfikować dla wartości wyjścia, wartości wewnętrznych, wartości zależnych od czasu (np. przed lub po jakimś zdarzeniu) oraz dla parametrów interfejsów (np. podczas testowania integracyjnego). Testy można tak zaprojektować, aby pokryć klasy równoważności. Podział na klasy równoważności (KR) daje się stosować na wszystkich poziomach testowania.

Klasy równoważności jako techniki można używać, aby osiągnąć pokrycie danych wejściowych i wyjściowych. Można ją również stosować w przypadku elementów wprowadzanych przez człowieka, elementów wprowadzanych do systemu poprzez interfejsy lub parametrów interfejsowych w testowaniu integracyjnym.

#### **4.3.2. Analiza wartości brzegowych (K3)**

Maksymalne i minimalne wartości klasy równoważności są jej wartościami brzegowymi. Istnieje możliwość, że zachowanie na krawędzi każdej klasy równoważności będzie nieprawidłowe, a więc brzegi są miejscem, gdzie testowanie prawdopodobnie wykryje defekty. Wartość brzegowa dla poprawnej klasy równoważności jest dozwoloną wartością brzegową; brzeg niepoprawnej klasy równoważności jest niepoprawną wartością brzegową. Testy mogą być projektowane w celu osiągnięcia pokrycia zarówno poprawnych jak i niepoprawnych wartości brzegowych. Projektując przypadki testowe wybierane są wszystkie wartości brzegowe.

Analizę wartości brzegowych można wykorzystywać na wszystkich poziomach testów. Jest ona łatwa w stosowaniu i posiada dużą zdolność znajdowania błędów. W technice tej pomocne są szczegółowe specyfikacje.

Technika wartości brzegowych często jest uwzględniana jako rozszerzenie techniki klas równoważności i może być wykorzystywana zarówno dla elementów wprowadzanych przez człowieka jak i dla elementów z zależnościami czasowymi lub elementów tablicowych. Wartości brzegowe mogą być również wykorzystane do określania danych testowych.

#### **4.3.3. Testowanie w oparciu o tablicę decyzyjną (K3)**

Tablice decyzyjne to dobry sposób na wychwycenie wymagań zawierających warunki logiczne i udokumentowanie wewnętrznego projektu systemu. Można ich użyć do zapisania złożonych reguł biznesowych, które system musi implementować. Analizuje się specyfikację, identyfikując warunki i akcje systemu. Warunki wejściowe i akcje ustawia się najczęściej w taki sposób, aby mogły odzwierciedlać prawdę lub fałsz (Boolean). Tablica decyzyjna zawiera warunki przełączania, często kombinacje prawdy i fałszu, dla wszystkich warunków wejściowych oraz wynikające z nich akcje. Każda kolumna tablicy odpowiada regule

biznesowej, która określa unikalną kombinację warunków, które w rezultacie dają wykonanie wszystkich akcji związanych z tą regułą. Podstawowym pokryciem wykorzystywanym w testowaniu w oparciu o tablice decyzyjne jest przynajmniej jeden test dla każdej z kolumn, który zwykle wymaga pokrycia wszystkich kombinacji warunków wyzwalających.

Siła testowania w oparciu o tablicę decyzyjną polega na tym, że tworzy się kombinacje warunków, które w inny sposób mogłyby nie być przetestowane. Testowanie to może być stosowane we wszystkich sytuacjach, kiedy działanie oprogramowania zależy od kilku decyzji logicznych.

#### **4.3.4. Testowanie przejść pomiędzy stanami (K3)**

System może dawać różną odpowiedź w zależności od bieżących warunków lub jego historii (jego stanu). W takim przypadku system można przedstawić jako diagram przejść pomiędzy stanami. Pozwala to testerowi widzieć oprogramowanie w kategoriach jego stanów, przejść pomiędzy stanami, wejść lub zdarzeń, które wyzwalają zmiany stanów (przejścia) i akcji, które mogą wynikać z tych przejść. Stany systemu lub testowanego obiektu są rozdzielne, identyfikowalne i policzalne. Tablica stanów pokazuje związek pomiędzy stanami a wejściami i może wskazać na przejścia, które są niedozwolone. Testy mogą być projektowane w taki sposób, aby pokryć każdy stan, wykonać każde przejście, wykonać specyficzne sekwencje przejść lub przetestować przejścia niedozwolone.

Testowanie przejść pomiędzy stanami wykorzystywane jest w przemyśle tworzącym oprogramowanie wbudowane oraz w automatyce. Technika ta jest również odpowiednia dla modelowania obiektu biznesowego mającego specyficzne stany lub testowania przepływów okien dialogowych (np. dla aplikacji internetowych lub scenariuszy biznesowych).

#### **4.3.5. Testowanie w oparciu o przypadki użycia (K2)**

Testy mogą być specyfikowane w oparciu o scenariusze biznesowe lub przypadki użycia. Przypadek użycia opisuje interakcję pomiędzy aktorami (użytkownikami) i systemem, która produkuje wartość wynikową dla użytkownika. Każdy przypadek użycia posiada warunki początkowe, które muszą być spełnione, aby przypadek zakończył się powodzeniem. Każdy przypadek użycia kończy się warunkami końcowymi, które są wynikiem po jego wykonaniu i stanem końcowym systemu. Przypadek użycia ma scenariusz główny (najbardziej prawdopodobny) a czasami gałęzie alternatywne.

Przypadki użycia opisują „przepływy procesów” przez system, bazując na jego możliwym użyciu. Zatem przypadki testowe powstające z przypadków użycia są bardzo przydatne w wyszukiwaniu niewykrytych defektów w rzeczywistych przepływach procesów wykonywanych w systemie.

Przypadki użycia, mające związek ze scenariuszami, są bardzo użyteczne w projektowaniu testów akceptacyjnych z udziałem klienta/użytkownika. Pomagają również wykryć defekty integracji powodowane przez współpracę lub kolidowanie różnych modułów, a niewykryte przez indywidualne testy każdego z modułów.

#### **4.4. Techniki na podstawie struktury lub białoskrzynkowe (K3), 60 minut**

##### Terminologia

Pokrycie kodu, pokrycie decyzji, pokrycie instrukcji kodu, testowanie strukturalne, testowanie na podstawie struktury, testowanie białoskrzynkowe.

##### Wprowadzenie

Testowanie na podstawie struktury/testowanie białoskrzynkowe opiera się na zidentyfikowanej strukturze oprogramowania lub systemu, jak pokazano w poniższych przykładach:

- Poziom modułu: strukturą jest kod, tzn. instrukcje, decyzje i rozgałęzienia.
- Poziom integracji: strukturą może być drzewo wywołań (diagram, w którym moduły wołają inne moduły).
- Poziom systemu: strukturą może być struktura menu, proces biznesowy, struktura strony WWW.

W sekcji tej przedyskutowano dwie techniki strukturalne dla pokrycia kodu bazujące na instrukcjach i decyzjach. Dla testowania decyzyjnego, aby pokazać alternatywy każdej decyzji, można użyć diagramu przepływu sterowania.

##### **4.4.1. Testowanie instrukcji i pokrycie (K3)**

W testowaniu modułowym, pokrycie instrukcji kodu jest procentową oceną wykonywalnych instrukcji, które zostały wykonane przez zestaw przypadków testowych. W testowaniu instrukcji przypadki testowe projektuje się tak, aby wykonać określone instrukcje - zwiększyć pokrycie instrukcji kodu.

##### **4.4.2. Testowanie decyzyjne i pokrycie (K3)**

Pokrycie decyzji, w odniesieniu do testowania rozgałęzień, jest procentową oceną rezultatów decyzyjnych (opcji Prawda i Fałsz z instrukcji IF), które zostały wykonane przez zestaw przypadków testowych. W testowaniu decyzyjnym przypadki testowe projektuje się tak, aby uzyskać określone rezultaty decyzyjne – zwiększyć pokrycie decyzji.

Testowanie decyzyjne jest rodzajem testowania przepływu sterowania, ponieważ generuje określony przepływ sterowania przez punkty decyzyjne. Pokrycie decyzji jest silniejsze niż pokrycie instrukcji kodu: 100% pokrycie decyzji gwarantuje 100% pokrycie instrukcji kodu, ale nie odwrotnie.

##### **4.4.3. Inne techniki na podstawie struktury (K1)**

Istnieją lepsze poziomy pokrycia strukturalnego poza pokryciem decyzyjnym, np. pokrycie warunków i pokrycie warunków wielokrotnych.

Koncepcja pokrycia może być również stosowana na innych poziomach testów (np. na poziomie integracji), gdzie procent modułów lub klas, które zostały sprawdzone przez zestaw przypadków testowych, może wyrażać pokrycie modułów lub klas.

W testowaniu strukturalnym przydatne jest wsparcie narzędziowe.

#### **4.5. Techniki oparte na doświadczeniu (K2), 30 minut**

##### Terminologia

Zgadywanie błędów, testowanie eksploracyjne.

##### Wprowadzenie

Najbardziej rozpowszechnioną techniką testowania jest zgadywanie błędów. Testy tworzone są na podstawie umiejętności, intuicji i doświadczenia testerów z podobnymi aplikacjami i technologiami. Testowanie intuicyjne, wykorzystane do uzupełnienia technik systematycznych, może być przydatne do wykonania specjalnych testów trudno wykonalnych przez techniki formalne (zwłaszcza, gdy stosowane jest po bardziej formalnych podejściach). Jednakże technika ta może znacznie różnić się efektywnością, zależną od doświadczenia testerów. Uporządkowanym podejściem dla techniki zgadywania błędów jest sporządzenie wykazu możliwych błędów i takie zaprojektowanie testów, aby je znaleźć. Wykazy defektów i awarii mogą być budowane na podstawie doświadczenia testerów, dostępnych danych o defektach i awariach oraz ogólnej wiedzy, dlaczego oprogramowanie ulega awariom.

Testowanie eksploracyjne jest równoczesnym projektowaniem, wykonywaniem, logowaniem testu oraz nauką, opartymi na karcie testu zawierającej cele testu i ramy czasowe ich wykonania. Podejście to jest użyteczne w przypadku niepełnej lub nieodpowiedniej specyfikacji, nacisków na skrócenie czasu testowania, lub dla poszerzenia i uzupełnienia bardziej formalnego testowania. Technika ta może służyć jako sprawdzenie procesu testowego, aby upewnić się, że najpoważniejsze defekty zostały wykryte.

#### **4.6. Wybór technik testowych (K2), 15 minut**

##### Terminologia

Brak szczególnych pojęć.

##### Wprowadzenie

Wybór technik testowych zależy od szeregu czynników, wliczając w to typ systemu, obowiązujące standardy, wymagania umowy lub klienta, poziom ryzyka, typ ryzyka, cel testu, dostępną dokumentację, doświadczenie testerów, czas i budżet, cykl życia projektu, modele przypadków użycia oraz wiedza o rodzajach znalezionych wcześniej błędów.

Pewne techniki dają się stosować dla określonych sytuacji i poziomów testów; inne można stosować na każdym poziomie testów.

##### Referencje:

4.1 Craig, 2002, Hetzel, 1998, IEEE 829

4.2 Beizer, 1990, Copeland, 2004

4.3.1 Copeland, 2004, Myers, 1979

4.3.2 Copeland, 2004, Myers, 1979

4.3.3 Beizer, 1990, Copeland, 2004

4.3.4 Beizer, 1990, Copeland, 2004

4.3.5 Copeland, 2004

4.4.3 Beizer, 1990, Copeland, 2004

4.5 Kaner, 2002

4.6 Beizer, 1990, Copeland, 2004

## 5. Zarządzanie testowaniem (K3), 180 minut

***Celem rozdziału „Zarządzanie testowaniem” jest nauczanie, w poszczególnych paragrafach:***

### 5.1 Organizacja testowania (K2)

- Znaczenia niezależnego testowania. (K1)
- Zalet i wad niezależnego testowania w organizacji. (K2)
- Jakie osoby powinny wchodzić w skład zespołu testowego. (K1)
- Typowych zadań kierownika testów i testera. (K1)

### 5.2 Planowanie i oszacowanie pracochłonności testów (K2)

- Różnych poziomów oraz celów planowania testów. (K1)
- Celu i zawartości planu testów, specyfikacji projektu testów oraz procedury testowej wg definicji z normy „Norma Dokumentacji Testowej” (Standard for Software Test Documentation, IEEE 829). (K2)
- Typowych czynników wpływających na wysiłek związany z testowaniem. (K1)
- Różnic w dwóch sposobach podejścia do oszacowania pracochłonności testowania: na podstawie danych pomiarowych oraz na podstawie oceny eksperckiej. (K2)
- Określenia zakresu planowania testów dla projektu, dla poszczególnych poziomów testów (np. dla testu systemowego), dla określonych celów (np. testy użyteczności) oraz dla wykonywania testów. (K2)
- Czynności wchodzących w skład przygotowywania i wykonywania testów, które wymagają planowania. (K1)
- Uzasadnienia odpowiednich warunków wyjścia dla określonych poziomów testów lub grup przypadków testowych (np. przypadków testowych dla testów integracyjnych, testów akceptacyjnych lub testów użyteczności). (K2)

### 5.3 Monitorowanie i nadzorowanie procesu testowego (K2)

- Jakie typowe metryki stosuje się do monitorowania przygotowywania i wykonywania testów.
- Interpretacji metryk dotyczących raportowania i nadzorowania testów (np. liczba znalezionych i naprawionych błędów, wykonanych zaliczonych i niezaliczonych testów). (K2)
- Celu i treści końcowego raportu testowego wg definicji z normy „Norma Dokumentacji Testowej” (Standard for Software Test Documentation, IEEE 829). (K2)

### 5.4 Zarządzanie konfiguracją (K2)

- W jaki sposób zarządzanie konfiguracją wspiera testowanie.

### 5.5 Ryzyko a testowanie (K2)

- Ryzyka jako możliwego problemu, zagrażającego osiągnięciu celów jednego lub wielu interesariuszy projektu. (K2)
- Że ryzyko określa prawdopodobieństwo wystąpienia zdarzenia oraz jego konsekwencje (koszty wystąpienia ryzyka). (K1)
- Różnic między ryzykami projektowymi oraz ryzykami odnoszącymi się do produktu.
- Rozpoznawania typowych zagrożeń dotyczących produktu i projektu.
- Jak analiza i planowanie ryzyka mogą być wykorzystane do planowania testów. (K2)

### 5.6 Zarządzanie incydentami (K3)

- Treści zgłoszenia incydentu wg normy „Norma Dokumentacji Testowej” (Standard for Software Test Documentation, IEEE 829). (K1)
- Pisania zgłoszenia incydentu dotyczącego awarii zaobserwowanej w trakcie testowania. (K3)

## 5.1. Organizacja testowania (K2), 30 minut

### Terminologia

Tester, kierownik testów, menedżer testów.

#### 5.1.1. Organizacja i niezależność testowania (K2)

Skuteczność znajdowania błędów drogą testowania i przeglądów można podnieść wykorzystując niezależnych testerów. Istnieją następujące formy niezależnego testowania:

- Niezależni testerzy w zespołach programistów.
- Niezależny zespół albo grupa testowa w organizacji, podlegająca kierownictwu projektu albo kierownictwu przedsiębiorstwa.
- Niezależni testerzy z działu biznesowego, przedstawiciele użytkowników końcowych oraz działu informatyki.
- Niezależni specjaliści testowi dla określonych celów, tacy jak testerzy użyteczności, testerzy zabezpieczeń lub testerzy certyfikacyjni, testujący oprogramowanie pod kątem zgodności z normami i innymi regulacjami prawnymi.

Dla dużych, złożonych projektów lub w zespołach wytwarzających produkty krytyczne pod względem bezpieczeństwa, zwykle najkorzystniejsze jest testowanie na wielu poziomach, gdzie pewne poziomy testowania wykonywane są przez niezależnych testerów. Deweloperzy mogą uczestniczyć w testowaniu, zwłaszcza na niższych poziomach, ale ich brak obiektywności ogranicza ich skuteczność. Niezależni testerzy mogą uzyskać upoważnienie do domagania się oraz definiowania procesów i procedur testowych, ale tego rodzaju żądania dotyczące procesu powinny być przez nich podejmowane tylko pod warunkiem uzyskania jednoznacznego upoważnienia ze strony kierownictwa.

Korzyści niezależności to:

- Niezależni testerzy dostrzegają często inne błędy niż dostrzeżone przez konstruktorów, a ponadto podchodzą do testowanego oprogramowania bez gotowych oczekiwań i uprzedzeń.
- Niezależny tester może także zweryfikować poprawność założeń, których dokonano podczas specyfikacji oraz implementacji systemu.

Wady niezależności to:

- Odizolowanie testerów od zespołu deweloperów (gdy niezależność jest zupełna).
- Niezależni testerzy mogą stać się wąskim gardłem projektu, gdy spoczywa na nich odpowiedzialność za ostateczną kontrolę produktu.
- Deweloperzy mogą utracić poczucie współodpowiedzialności za jakość.

Zadania testowe bywają wykonywane przez osoby mające przypisaną rolę testerów, lub przez inne osoby, na przykład przez kierownika projektu, kierownika jakości, dewelopera, specjalistę biznesowego i dziedzinowego, eksperta z działu informatyki lub odpowiedzialnego za infrastrukturę.

#### 5.1.2. Zadania kierownika testów i testera (K1)

Niniejszy plan omawia dwie role związane z testowaniem: kierownika testów oraz testera. Czynności i zadania wykonywane przez osoby wykonujące te dwie role zależą od formy projektu i rodzaju produktu, od osób realizujących te role oraz od specyfiki organizacyjnej.

Czasem kierownik testów bywa nazywany menedżerem lub koordynatorem testów. Funkcję kierownika testów może wykonywać kierownik projektu, kierownik zespołu programistów, kierownik jakości albo menedżer zespołu testowego. W dużych projektach mogą

występować obie role: kierownika i menedżera testów. Kierownik testów zwykle planuje, monitoruje i nadzoruje czynności i zadania testowe w sposób opisany w części 1.4.

Zwykle spotykane zadania kierownika testów to:

- Koordynowanie strategii i planu testów z kierownikami projektów i innymi interesariuszami.
- Stworzenie lub dokonanie przeglądu strategii testowej dla projektu, a polityki testowej dla organizacji.
- Dbłość o uwzględnienie aspektów testowych w innych obszarach projektu, np. w planowaniu integracji.
- Planowanie testowania – z uwzględnieniem kontekstu i ryzyka. Planowanie obejmuje wybór metodyk testowania, oszacowanie czasochłonności, wysiłku oraz kosztów testowania, uzyskanie odpowiednich zasobów, określenie poziomów, cykli, metod oraz celów testowania, a także zaplanowanie zarządzania zgłoszeniami zdarzeń.
- Zainicjowanie i nadzorowanie specyfikacji, przygotowania oraz implementacji testów, a w szczególności monitorowanie i nadzorowanie ich wykonywania,
- Ustawienie odpowiedniego poziomu zarządzania konfiguracją,
- Wprowadzenie odpowiednich metryk, by mierzyć postęp testowania i określać jakość testowania i produktu
- Określenie zakresu i stopnia automatyzacji.
- Wybór narzędzi wspierających testowanie oraz zorganizowanie niezbędnych szkoleń narzędziowych.
- Podjęcie decyzji dotyczących środowiska testowego.
- Sporządzenie harmonogramu testowania.
- Sporządzanie końcowych raportów testowych na podstawie informacji zgromadzonych podczas wykonywania testów.

Typowe zadania testera obejmują:

- Przeglądanie i wnoszenie wkładu do planu testów
- Analiza, przegląd i dostęp do wymagań użytkownika, specyfikacji oraz modeli testowalności
- Tworzenie środowiska testowego (często we współpracy z administracją systemu i zarządzającymi siecią)
- Przygotowanie i pozyskiwanie danych testowych
- Implementacja testów na wszystkich poziomach, wykonywanie testów, zapisywanie wyników do dziennika (logu) testów, porównywanie wyników i dokumentowanie odchyleń od spodziewanego wyniku
- Używanie – w razie potrzeby - narzędzi do administrowania lub zarządzania testami,
- Automatyzacja testów (w razie potrzeby we współpracy z deweloperami lub ekspertami od automatyzacji testów)
- Mierzenie wydajności modułów lub systemów – jeśli ma to zastosowanie
- Przeglądanie testów zaprojektowanych przez innych

Osoby, które pracują przy analizie testów, projektowaniu testów, testach specyficznych typów lub automatyzacji testów powinny być specjalistami w swych rolach. W zależności od poziomu testów i ryzyka związanego z produktem lub projektem, różni ludzie mogą pracować jako testerzy, zachowując pewien stopień niezależności. Na ogół, na poziomie modułów i integracji, testerami mogą być deweloperzy, przy testach akceptacyjnych mogą to być eksperci biznesowi lub użytkownicy, przy akceptacji operacyjnej testerami powinni być operatorzy systemu.

## 5.2. Planowanie testowania (K2), 50 minut

### Terminologia

Kryteria wejścia, kryteria wyjścia, testowanie eksploracyjne, metodyka testowa, poziom testowy, plan testów, procedura testów, strategia testowa.

#### 5.2.1. Planowanie testowania (K2)

Niniejsza sekcja poświęcona jest planowaniu testowania w projektach wytwarzania, implementacji oraz utrzymania oprogramowania. Planowanie może być udokumentowane w planie testów projektu lub w głównym planie testów, a także w odrębnych planach testów dla poszczególnych poziomów, np. dla testu systemowego lub testu akceptacyjnego. Wzorce dokumentacji planów testów znajdują się w *Standard for Software Test Documentation* (IEEE 829).

Na planowanie wpływa: polityka testów w organizacji, zakres testowania, cele, zagrożenia, ograniczenia, krytyczność, łatwość testowania i dostępność zasobów. Im dalej posuwa się planowanie projektu i testowania, tym więcej informacji jest do dyspozycji i tym bardziej szczegółowy staje się plan.

Planowanie jest stałą czynnością i wykonuje się je we wszystkich procesach i fazach cyklu życiowego oprogramowania. Informacja zwrotna z realizacji planu testów umożliwia identyfikację zmieniającego się ryzyka, co pozwala na dostosowywanie planu.

#### 5.2.2. Czynności wykonywane podczas planowania testów (K2)

W trakcie planowania testów mogą być wykonywane następujące czynności:

- Określenia ogólnej metodyki (strategii) testowej, w tym poziomów testowania oraz kryteriów wejścia (dopuszczenia do testowania) i wyjścia (zakończenia testowania).
- Zintegrowanie i skoordynowanie testowania z fazami cyklu życiowego oprogramowania: zakup, dostawa, wytwarzanie, działanie operacyjne oraz utrzymanie.
- Podejmowanie decyzji, co będzie przedmiotem testów, jakie role będą przypisane, jakim zadaniom, kiedy i w jaki sposób wykonywane będą zadania testowe, jak będą oceniane wyniki testów, kiedy zakończyć testowanie (kryteria wyjścia).
- Przydzielanie zasobów zdefiniowanym zadaniom.
- Określenie ilości, szczegółowości, struktury i wzorców dokumentacji testowej.
- Wybór miar służących do monitorowania i nadzoru nad przygotowaniem i wykonywaniem testów, rozwiązywania zgłoszeń błędów oraz zagadnień związanych z ryzykiem.
- Określenie szczegółowości procedur testowych w stopniu dostatecznym dla potrzeb wielokrotnego przygotowywania i wykonywania testów.

#### 5.2.3. Kryteria wyjścia (K2)

Kryteria wyjścia określają, kiedy można zakończyć testowanie, na przykład po wykonaniu testów na danym poziomie lub po wykonaniu zestawów testów mających określony cel.

Przykłady powszechnie stosowanych kryteriów zakończenia:

- Miary staranności takie jak pokrycie kodu, funkcjonalności lub ryzyka.
- Oszacowania zagęszczenia błędów lub miary niezawodności.
- Koszty.

- Istniejące zagrożenia, takie jak nienaprawione błędy lub brak pokrycia pewnych obszarów.
- Harmonogramy, na przykład narzucające datę dostawy.

#### **5.2.4. Oszacowanie wysiłku testowego (K2)**

Plan omawia dwa sposoby oszacowywania wysiłku testowego:

- Oszacowanie wysiłku testowego na podstawie pomiarów z wcześniejszych lub z podobnych projektów albo na podstawie typowych wielkości.
- Oszacowanie zadań przez osoby za nie odpowiedzialne lub przez ekspertów.

Po oszacowaniu pracochłonności zadań możliwe jest przypisanie im zasobów i sporządzenie harmonogramu.

Pracochłonność testowania zależy od wielu czynników, w tym:

Właściwości produktu: jakości specyfikacji i innych źródeł informacji, na podstawie, których budowane są modele (np. podstawy testowej), wielkości produktu, złożoności zagadnień dziedziny, wymagań dotyczących niezawodności i zabezpieczeń oraz wymagań dotyczących dokumentacji.

Właściwości procesu wytwarzania: stabilności organizacji, stosowanych narzędzi, procesu testowego, umiejętności personelu, napiętości harmonogramu.

Wyników testowania: liczby znalezionych błędów i ilości niezbędnych przeróbek.

#### **5.2.5. Sposoby podejścia do testowania (strategie testowe) (K2)**

W Jednym ze sposobów kategoryzacji podejścia lub strategii testowych stosuje się kryterium czasu, kiedy odbywa się większość pracy związanej z testowaniem:

- Metody prewencyjne, w których testy projektuje się tak wcześnie jak to tylko możliwe.
- Metody reaktywne, w których projektowanie testów następuje dopiero po wyprodukowaniu oprogramowania lub systemu.

Typowe strategie testowe to:

- Metody analityczne, na przykład testowanie na podstawie ryzyka, w których testowanie koncentrowane jest w obszarach najwyższego zagrożenia.
- Metody modelowania, na przykład testowanie stochastyczne na podstawie danych statystycznych dotyczących częstotliwości awarii (modele wzrostu niezawodności) lub dotyczących częstotliwości zastosowania (profile użycia operacyjnego).
- Metody systematyczne, takie jak testowanie na podstawie danych o awariach (w tym domyślanie się błędów i ataki przy pomocy błędnych danych), na podstawie list kontrolnych oraz na podstawie atrybutów jakości.
- Metody zgodne z procesem, standardem lub normą, na przykład określone przez standard branżowy lub metodyki zwinne.
- Metody dynamiczne i heurystyczne, na przykład testowanie eksploracyjne, w których testowanie polega bardziej na reagowaniu na wydarzenia niż na planowaniu, i gdzie wykonywanie i ocena odbywają się równolegle.
- Metodyki konsultatywne, w których projektowanie testów jest w przeważającej mierze zadaniem ekspertów w dziedzinie technologii lub zastosowania biznesowego, nienależących do zespołu testowego.
- Podejścia mające na celu ograniczenie pracochłonności testów regresji poprzez ponowne użycie artefaktów testowych, rozbudowaną automatyzację oraz wykorzystanie standardowych zestawów testów.

Różne metody można łączyć, np. dynamiczna metodyka na podstawie oszacowania ryzyka.

Wybierając metodykę testową, należy uwzględnić takie aspekty jak:

- Ryzyko niepowodzenia projektu, zagrożenia wobec produktu oraz zagrożenie dla ludzi, środowiska lub firmy spowodowane awarią produktu.
- Umiejętności i doświadczenie ludzi uczestniczących w projekcie w zakresie planowanych technik, narzędzi i metod.
- Cel przedsięwzięcia testowego oraz misję zespołu testowego.
- Przepisy, zarówno zewnętrzne jak i wewnętrzne, dotyczące procesu wytwarzania.
- Rodzaj i specyfikę produktu lub przedsięwzięcia.

### **5.3. Monitorowanie przebiegu i nadzór testowania (K2), 20 minut**

#### Terminologia

Gęstość defektów, częstotliwość awarii, nadzór nad testowaniem, pokrycie testowe, monitorowanie testowania, raport testowy.

#### **5.3.1. Monitorowanie postępu testów (K1)**

Celem monitorowania testów jest uzyskanie informacji zwrotnej o zadaniach testowych oraz unaocznienie ich przebiegu. Monitorowaną informację można zbierać ręcznie lub automatycznie i można ją stosować do pomiaru spełnienia kryteriów wyjścia, (np. analizy pokrycia). Pomiarów mogą służyć również do oceny postępu prac w porównaniu z harmonogramem i budżetem. Przykłady często stosowanych miar:

- Jaka część prac przygotowawczych została wykonana, lub jaki odsetek zaplanowanych przypadków testowych przygotowano (wykonano).
- Jaką część wykonano przygotowując środowisko testowe.
- Przebieg wykonania testów (np. liczba wykonanych i niewykonanych przypadków testowych, liczba zaliczonych i niezaliczonych przypadków testowych).
- Dane o błędach, np. gęstość defektów, liczba znalezionych i naprawionych defektów, częstotliwość awarii, wyniki testów zgodności lub retestów.
- Pokrycie wymagań, ryzyka lub kodu.
- Subiektywne poczucie testerów dotyczące niezawodności produktu.
- Daty testowych kamieni milowych.
- Koszty testowania, w tym koszty porównawcze wobec korzyści znalezienia następnego defektu lub wykonania kolejnego testu.

#### **5.3.2. Raportowanie testów (K2)**

Celem raportowania jest podanie podsumowujących danych dotyczących przebiegu przedsięwzięcia testowego, takich jak:

- Co zdarzyło się w okresie testowania, np. daty spełnienia kryteriów wyjścia.
- Analiza danych oraz pomiarów w celu uzasadnienia rekomendacji i decyzji, np. ocena ile pozostało jeszcze defektów, ekonomiczna opłacalność dalszego testowania, istniejące zagrożenia, oraz określenie poziomu zaufania do testowanego oprogramowania.

Przykładowy wzorzec końcowego raportu testowego znajduje się w normie IEEE 829 „Standard for Software Test Documentation”.

W trakcie i pod koniec każdego poziomu testów należy wykonywać pomiary, aby móc ocenić następujące elementy:

- Właściwy dobór celów testowania dla danego poziomu.
- Na ile przyjęta metodyka testowa jest odpowiednia.
- Na ile testowanie jest skuteczne w porównaniu z przyjętymi celami.

#### **5.3.3. Nadzór nad testowaniem (K2)**

Nadzór nad testowaniem obejmuje wszelkie czynności nadające testom kierunek oraz wszelkie działania naprawcze, które podejmuje się w wyniku zgromadzonych i zaraportowanych danych i pomiarów. Czynności te mogą dotyczyć wszystkich zadań testowych i mogą wywierać wpływ na wszystkie inne czynności podejmowane w cyklu życia oprogramowania.

Przykłady czynności nadzorczych:

- Zmiana wag testów po zmaterializowaniu się jakiegoś ryzyka (np. spóźnionej dostawie oprogramowania).
- Zmiana harmonogramu uwzględniająca dostępność środowiska testowego.
- Ustalenie kryteriów wejściowych wymagających, aby naprawy błędów były ponownie przetestowane przez programistę przed ich włączeniem do budowy nowej wersji oprogramowania.

#### 5.4. Zarządzanie konfiguracją (K2), 10 minut

##### Terminologia

Zarządzanie konfiguracją, nadzór nad wersjami.

##### Wprowadzenie

Celem zarządzania konfiguracją jest uzyskanie i utrzymanie spójności produktów (komponentów, danych i dokumentacji) oprogramowania lub systemu podczas projektu i w cyklu życiowym produktu.

Z punktu widzenia testowania, zarządzanie konfiguracją służy do osiągnięcia:

- Identyfikacji wersji, nadzoru nad nimi, zapisywania zmian, śledzenia relacji między artefaktami testowymi oraz relacji między nimi a przedmiotem testu w celu utrzymania kontroli nad przeprowadzonymi zmianami w trakcie całego procesu testowania.
- Jednoznacznej identyfikacji wszelkich dokumentów oraz elementów oprogramowania, do których odwołuje się dokumentacja testowa.

Zarządzanie konfiguracją umożliwia testerom jednoznaczne zidentyfikowanie i odtworzenie testowanego elementu, dokumentacji testowej, testów oraz narzędzia testowego.

Planowanie testów powinno określić, udokumentować i wdrożyć procedury oraz infrastrukturę (narzędzia) do zarządzania konfiguracją.

## 5.5. Ryzyko a testowanie (K2), 30 minut

### Terminologia

Ryzyko produktowe, ryzyko projektowe, testowanie na podstawie ryzyka.

### Wprowadzenie

Ryzyko można zdefiniować jako prawdopodobieństwo wystąpienia wydarzenia, zagrożenia lub niepożądanego w skutkach sytuacji jako możliwego problemu. Poziom ryzyka określają prawdopodobieństwo niekorzystnego zdarzenia oraz jego konsekwencje (szkoda będąca skutkiem tego zdarzenia).

#### 5.5.1. Ryzyko projektowe (K1, K2)

Ryzyko projektowe to ryzyko zdolności projektu do osiągnięcia celów, na przykład:

- Problemy z dostawcami:
- Niepowodzenie dostawy;
- Kwestie dotyczące kontraktu.
- Czynniki organizacyjne:
  - Brak personelu i/lub niezbędnych umiejętności
  - Problemy osobiste i problemy dotyczące szkoleń;
- Kwestie organizacyjne, np.
  - Nieskuteczne komunikowanie przez testerów swoich potrzeb oraz wyników testów;
  - Niewykorzystanie informacji zdobytej podczas testowania i przeglądów, np. zaniechanie usprawniania procedur i innych praktyk testowych).
  - Niewłaściwa postawa lub oczekiwania w stosunku do testów (niedocenianie wyszukiwania defektów podczas testowania).
- Czynniki techniczne:
  - Niewłaściwie określone wymagania;
  - Wymagania nie są w pełni osiągalne przy istniejących ograniczeniach;
  - Jakość projektu, kodu i testów.

Analizując, zarządzając i zapobiegając tym zagrożeniom, kierownik projektu postępuje według znanych zasad zarządzania projektami. Wzorzec planu testów wg normy IEEE 829 („Standard for Software Test Documentation”) wymaga określenia zagrożeń i zaplanowania czynności zaradczych.

#### 5.5.2. Ryzyko związane z produktem (K2)

Jako ryzyko związane z produktem określa się obszary możliwych awarii (niekorzystne przyszłe wydarzenia lub zagrożenia) w oprogramowaniu lub systemie, gdyż zagrażają one w różny sposób jakości produktu, np.:

- Przez dostarczenie zawodnego oprogramowania.
- Oprogramowanie lub sprzęt spowodują szkody osobiste i/lub dla firmy.
- Niedostateczne właściwości oprogramowania (np. funkcjonalność, zabezpieczenia, niezawodność, użyteczność i wydajność).

- Oprogramowanie, które nie spełnia założonej funkcjonalności.

Przy pomocy ryzyka można określić, kiedy należy rozpocząć testowanie i gdzie warto testować więcej. Testowanie ogranicza prawdopodobieństwo niekorzystnego wydarzenia lub zmniejsza skutki jego wystąpienia.

Ryzyko związane z produktem jest rodzajem zagrożenia dla powodzenia projektu. Testowanie pozwala nadzorować ryzyko, dostarczając informacji o poziomie zagrożenia dzięki pomiarom skuteczności usuwania krytycznych defektów oraz planów awaryjnych.

Metodyka testowania bazująca na ryzyku stwarza możliwości aktywnego ograniczania ryzyk związanych z produktem od najwcześniejszych etapów projektu. Testowanie umożliwia identyfikację ryzyk, a następnie wykorzystania ich w planowaniu, specyfikacji, przygotowaniu i wykonaniu testów. Zidentyfikowane ryzyka można wykorzystać do:

- Określenia, jakie zastosować techniki testowania.
- Określenia zakresu testów.
- Uporządkowania testów pod względem ich wag, aby móc jak najwcześniej znaleźć najistotniejsze defekty.
- Określenia, jakie czynności poza testowaniem warto wykorzystać dla ograniczenia ryzyka (np. odpowiednio szkoleń niedoświadczonych projektantów).

Testowanie na podstawie ryzyka wykorzystuje wspólną wiedzę i doświadczenie interesariuszy w celu określenia ryzyk i poziomu testów niezbędnych dla ich kontrolowania.

Aby zminimalizować ryzyko awarii produktu, w zarządzaniu ryzykiem stosuje się zdyscyplinowane metody mające na celu:

- Ocenę – regularnie powtarzaną – możliwych wystąpień niekorzystnych wydarzeń (ryzyk).
- Określenie wagi ryzyk.
- Wdrożenie czynności mających na celu eliminowanie ryzyk.

Testowanie umożliwia także identyfikację nowych ryzyk, określenie którym ryzykom należy zapobiegać oraz pozwala na trafniejszą ocenę ich prawdopodobieństwa.

## 5.6. Zarządzanie incydentami (K3), 40 minut

### Terminologia

Logowanie incydentów.

### Wprowadzenie

Jednym z celów testowania jest znajdowanie defektów, a więc należy prowadzić logowanie incydentów - niezgodności między oczekiwanymi a rzeczywistymi wynikami testów.

Incydenty należy nadzorować od ich odkrycia, poprzez zaklasyfikowanie, naprawienie i potwierdzenie skuteczności naprawy. Aby zarządzać incydentami, organizacje powinny określić i stosować proces i reguły ich klasyfikacji.

Incydenty mogą być zgłaszane podczas wytwarzania, przeglądów, testowania lub użytkowania oprogramowania. Incydenty mogą dotyczyć kodu, działającego systemu lub wszelkiej dokumentacji, w tym dokumentacji dotyczącej wytwarzania, dokumentacji testowej lub dokumentacji przeznaczonej dla użytkownika, takiej jak instrukcja użytkownika lub instrukcja instalacji.

Celem zgłoszeń incydentów jest:

- Dostarczenie konstruktorom i innym interesariuszom informacji o incydencie, by umożliwić identyfikację, wyizolowanie oraz – w razie potrzeby – naprawę błędu.
- Dostarczenie kierownictwu testów bieżącej informacji o jakości testowanego systemu oraz o przebiegu testów.
- Generowanie propozycji udoskonalenia procesu testowego.

Tester lub osoba wykonująca przegląd zwykle notuje następujące dane (gdy są dostępne) na temat incydentu:

- Datę zgłoszenia, organizację zgłaszającą, autora, zatwierdzenie i status.
- Czego dotyczy zgłoszenie, jego wagę i priorytet.
- Odnośniki, w tym identyfikator przypadku testowego, który ujawnił problem.

Zgłoszenie incydentu może zawierać następujące dane:

- Wynik rzeczywisty i oczekiwany.
- Datę odkrycia incydentu.
- Określenie elementu konfiguracji oprogramowania lub systemu.
- W jakiej fazie cyklu życia oprogramowania zaobserwowano incydent.
- Opis niezgodności ułatwiający określenie jej przyczyny.
- Konsekwencje dla interesariuszy.
- Określenie stopnia pilności rozwiązania.
- Status zgłoszenia (np. otwarte, odrzucone, powtórne zgłoszenie, oczekujące na naprawę, naprawione oczekujące na re-test, zamknięte).
- Wnioski i zalecenia.
- Zagadnienia ogólne, na przykład inne obszary, na które mogą mieć wpływ zmiany przeprowadzone skutkiem naprawy błędu powodującego incydent.

- Historia zmian, na przykład, jakie czynności zostały wykonane w celu wyizolowania, naprawienia oraz potwierdzenia skuteczności naprawy.

IEEE 829 („Standard for Software Test Documentation”) określa treść i strukturę zgłoszenia incydentu, nazywanego w normie tej zgłoszeniem niezgodności.

#### **Bibliografia:**

5.1.1 Black, 2001, Hetzel, 1998

5.1.2 Black, 2001, Hetzel, 1998

5.2.5 Black, 2001, Craig, 2002, IEEE 829, Kaner 2002

5.3.3 Black, 2001, Craig, 2002, Hetzel, 1998, IEEE 829

5.4 Craig, 2002

5.5.2 Black, 2001, IEEE 829

5.6 Black, 2001, IEEE 829

## **6. Testowanie wspierane narzędziami (K2), 80 minut**

***Celem rozdziału „Testowanie wspierane narzędziami” jest nauczanie, w poszczególnych paragrafach:***

### **6.1 Rodzaje narzędzi testowych (K2)**

- O różnych rodzajach narzędzi testowych nadających się do różnych czynności testowych. (K2)
- O narzędziach, które mogą być wykorzystane do testowania wykonywanego przez programistów. (K1)

### **6.2 Skuteczne użycie narzędzi: potencjalne korzyści i zagrożenia (K2)**

- Potencjalnych korzyści i zagrożeń automatyzacji testów oraz wykorzystania narzędzi wspierających testowanie. (K2)
- Różnych form pisania skryptów sterujących narzędziami wspierającymi wykonanie testów, w oparciu o dane i w oparciu o słowa kluczowe. (K1)

### **6.3 Wdrożenie narzędzi w organizacji (K1)**

- Zasad wdrażania narzędzi w organizacji. (K1)
- Celów fazy pilotażowej oceny narzędzia. (K1)
- Zasady, że dla uzyskania korzyści z użycia narzędzia nie wystarcza samo nabycie narzędzia. (K1)

## 6.1. Rodzaje narzędzi testowych (K2), 45 minut

### Terminologia

Narzędzie do zarządzania konfiguracją, narzędzie do pomiaru pokrycia, debager, sterownik, narzędzie do analizy dynamicznej, narzędzie do zarządzania zgłoszeniami błędów, narzędzie do testów obciążeniowych, narzędzie do modelowania, narzędzie monitorujące, narzędzie do testów wydajnościowych, efekt próbnika, narzędzie do zarządzania wymaganiami, narzędzie wspierające proces przeglądu, narzędzie do testów zabezpieczeń, narzędzie do analizy statycznej, narzędzie do testów przeciążenia, zaślepka, komparator, narzędzie do tworzenia danych testowych, narzędzie do projektowania testów, jarzmo testowe, narzędzie wspierające wykonanie testu, narzędzie wspierające zarządzanie testami, narzędzie do testów jednostkowych.

#### 6.1.1. Klasyfikacja narzędzi testowych

Istnieje wiele narzędzi wspierających różne aspekty testowania. Klasyfikacja zastosowana w niniejszym konspekcie dokonana jest na podstawie czynności testowych, które są wspierane przez narzędzia.

Niektóre narzędzia służą do wsparcia tylko jednej czynności, inne mogą być wykorzystywane do różnych czynności, ale zaklasyfikowane zostały zgodnie z tą czynnością, do której najczęściej są stosowane. Niektóre narzędzia komercyjne wspierają tylko jeden rodzaj czynności, ale niektórzy dostawcy komercyjnych narzędzi oferują zestawy albo rodziny narzędzi, wspierających wiele różnych czynności testowych.

Narzędzia umożliwiają usprawnienie testowania dzięki zautomatyzowaniu wielokrotnie wykonywanych czynności testowych. Narzędzia mogą także poprawić niezawodność testowania dzięki automatyzacji porównywania dużej ilości danych lub symulowania określonych zachowań środowiska testowanego systemu.

Niektóre rodzaje narzędzi są inwazyjne, co oznacza, że samo zastosowanie narzędzia wywiera wpływ na rzeczywisty wynik testu. Na przykład, rzeczywisty czas wykonania może zależeć od tego, w jaki sposób narzędzia do testów wydajnościowych dokonują pomiarów. Zależnie od zastosowanego narzędzia można otrzymać także różne wyniki pomiarów pokrycia kodu. Skutek inwazyjności narzędzia nazywany jest efektem próbnika.

Niektóre narzędzia wspierają czynności wykonywane najczęściej przez programistów (np. podczas testowania modułowego oraz testowania integracji modułów). Takie narzędzia oznaczone są literą „(D)” w poniższej klasyfikacji.

#### 6.1.2. Zarządzanie testowaniem wspierane narzędziami (K1)

Zarządzanie testowaniem wspierane narzędziami znajduje zastosowanie we wszystkich czynnościach testowych podczas całego cyklu życia oprogramowania.

### Narzędzia do zarządzania testami

Narzędzia do zarządzania testami mają następujące funkcje:

- Wsparcie zarządzania testami i innymi wykonywanymi czynnościami testowymi.
- Interfejsy do narzędzi wspierających wykonanie testu, zarządzanie zgłoszeniami błędów i zarządzanie wymaganiami.
- Wbudowane zarządzanie wersjami lub interfejs do zewnętrznego narzędzia do zarządzania wersjami.

- Możliwość śledzenia powiązań testów, wyników testów oraz incydentów z dokumentami źródłowymi, takimi jak specyfikacje wymagań.
- Logowanie wyników testów i tworzenie raportów z postępu realizacji testów.
- Analizę ilościową (miary) dotyczącą testów (np. liczba wykonanych i zaliczonych testów) oraz przedmiotu testów (np. liczba incydentów), mającą na celu uzyskanie informacji o przedmiocie testów oraz nadzorowanie i udoskonalanie procesu testowania.

## Narzędzia do zarządzania wymaganiami

Narzędzia do zarządzania wymaganiami służą do przechowywania wymagań, kontroli spójności wymagań oraz identyfikacji niezdefiniowanych (brakujących) wymagań. Narzędzia te umożliwiają ponadto określenie priorytetów wymagań, oraz śledzenie powiązań pojedynczych testów z wymaganiami, funkcjami lub obszarami funkcjonalności systemu. Powiązania te mogą być wykorzystywane w raportach postępu prac. Raportowane może być także pokrycie testami: wymagań, funkcji lub funkcjonalności systemu.

## Narzędzia do zarządzania incydentami

Narzędzia do zarządzania incydentami służą do przechowywania i zarządzania zgłoszeniami incydentów, czyli defektów, awarii oraz innych obserwowanych problemów lub anomalii. Wspierają one zarządzanie zgłoszeniami incydentów m.in. poprzez:

- Ułatwianie priorytetyzacji zgłoszeń.
- Przydzielanie osobom zadań do wykonania (np. naprawę błędu lub testowanie potwierdzające).
- Określenie statusu zgłoszenia (np. odrzucone, gotowe do przetestowania lub odłożone do następnego wypuszczenia wersji).

Narzędzia te umożliwiają śledzenie zmian treści i statusu incydentów, a także analizę statyczną oraz raportowanie incydentów. Nazywane są także narzędziami do śledzenia zgłoszeń błędów.

## Narzędzia do zarządzania konfiguracją

Narzędzia do zarządzania konfiguracją nie są w pełni narzędziami testowymi, ale są z reguły niezbędne do zarządzania różnymi wersjami i build'ami oprogramowania i testów.

Narzędzia do zarządzania konfiguracją:

- Przechowują dane na temat wersji i build'ów oprogramowania oraz artefaktów testowych (testaliów).
- Pozwalają śledzić powiązania między testaliami oraz produktami programistycznymi i ich wariantami.
- Są szczególnie przydatne, gdy wytwarzana jest więcej niż jedna konfiguracja oprogramowania i sprzętu (np. dla różnych wersji systemu operacyjnego, bibliotek lub kompilatorów, różnych przeglądark lub różnych typów komputerów).

### 6.1.3. Testowanie statyczne wspierane narzędziami (K1)

#### Narzędzia wspierające proces przeglądu

Narzędzia wspierające proces przeglądu mogą gromadzić dane na temat procesów przeglądu, przechowywać i służyć do przekazywania komentarzy przeglądowych, raportować znalezione defekty oraz pracochłonność, zarządzać odwołaniami do reguł wykonywania przeglądów lub list kontrolnych, a także śledzić powiązania między dokumentacją a kodem źródłowym. Mogą także pozwalać na wykonywanie przeglądów on-line, co jest pożądane w geograficznie rozproszonych projektach.

#### Narzędzia do analizy statycznej (D)

Narzędzia do analizy statycznej umożliwiają programistom, testerom i specjalistom od zapewnienia jakości znajdowanie defektów przed rozpoczęciem testowania dynamicznego. Ich podstawowe cele to:

- Nadzorowanie przestrzegania standardów kodowania.
- Analiza struktur i zależności (np. połączonych stron internetowych).
- Ułatwienie zrozumienia kodu.

Narzędzia do analizy statycznej na podstawie kodu mogą wyliczać miary (np. złożoność), co jest cenną informacją np. przy planowaniu i analizie ryzyka.

#### Narzędzia do modelowania (D)

Narzędzia do modelowania pozwalają dokonywać walidacji modeli oprogramowania. Na przykład sprawdzenie modelu bazy danych może znaleźć defekty i niespójności w modelu danych; inne narzędzia do modelowania pozwalają znajdować defekty w modelach przejść stanów (automatów skończonych) lub w modelach obiektowych. Często narzędzia tego rodzaju umożliwiają wygenerowanie przypadków testowych z modelu (patrz też „Narzędzia wspierające projektowanie testów” poniżej).

Podstawową korzyścią z zastosowania narzędzi do modelowania i analizy statycznej jest oszczędność wynikająca ze znajdowania błędów we wcześniejszych fazach procesu wytwarzania oprogramowania. Dzięki temu proces wytwarzania może odbywać się szybciej i sprawniej, gdyż ogranicza się ilość przeróbek.

### 6.1.4. Tworzenie specyfikacji testów wspierane narzędziami

#### Narzędzia wspierające projektowanie testów

Narzędzia wspierające projektowanie testów generują wejściowe dane testowe lub testy z wymagań, z graficznego interfejsu użytkownika, z modeli projektowych (stanów, danych lub obiektów) lub z kodu źródłowego. Ten rodzaj narzędzi pozwala także generować wyniki oczekiwane (np. wykorzystując wyrocznie). Testy wygenerowane z modelu automatu skończonego lub modelu obiektowego są przydatne do weryfikacji poprawności implementacji modelu w kodzie, ale z reguły nie są wystarczające dla zweryfikowania wszystkich aspektów oprogramowania lub systemu. Pozwalają zaoszczędzić cenny czas i zapewnić większą staranność dzięki temu, że testy wygenerowane przez narzędzia są kompletne i dotyczą wszystkich obszarów funkcjonalności systemu.

Inne narzędzia tego rodzaju wspomagają generowanie testów dostarczając strukturalizowanych wzorców, niekiedy nazywanych wzorcami testowymi, które generują testy lub zaślepki testowe, co przyspiesza proces projektowania testów.

## Narzędzia do przygotowywania danych testowych (D)

Narzędzia do przygotowywania danych testowych generują dane testowe wykorzystywane przy wykonywaniu testów z baz danych, plików lub zarejestrowanych przekazów danych. Zaletą tych narzędzi jest (dzięki usunięciu szczegółowych danych osobowych) zabezpieczenie poufnych danych wykorzystanych w środowisku testowym.

### 6.1.5. Wykonywanie testów wspierane narzędziami

#### Wykonywanie testów wspierane narzędziami

Wykonywanie testów wspierane narzędziami umożliwia automatyczne lub półautomatyczne wykonywanie testów sterowanych językiem skryptowym, wykorzystujących zebrane dane wejściowe oraz wyniki oczekiwane. Język skryptowy ułatwia sterowanie testami, na przykład wielokrotne powtórzenie testu z różnymi danymi lub przetestowanie różnych części systemu przy użyciu tych samych kroków. Zwykle narzędzia tego rodzaju pozwalają na dynamiczne porównywanie wyników oraz tworzenie logu dla każdego wykonania testu.

Narzędzia wspierające wykonanie testów mogą być też stosowane do rejestrowania testów. Rejestracja danych testowych wprowadzanych podczas testowania eksploracyjnego lub innego wykonywanego bez użycia skryptów testowych mogą być przydatne w celu odtworzenia lub udokumentowania testów w przypadku awarii.

#### Jarzma testowe oraz narzędzia wspomagające do testów jednostkowych

Jarzmo testowe ułatwia testowanie komponentów lub części systemu, symulując środowisko danego obiektu testowego. Jarzma testowe stosuje się z dwóch przyczyn. Po pierwsze, gdy inne komponenty środowiska nie są jeszcze dostępne i zastępuje się je zaślepkami lub sterownikami, a po drugie, aby wykonywanie testów odbywało się w sprawdzonym i przewidywalnym środowisku, co umożliwi lokalizację błędów w obiekcie testowym.

Rusztowanie testowe tworzy się, gdy testowany fragment kodu: obiekt, metoda, funkcja lub moduł jest wykonywany poprzez jego wywołanie i dostarczenie mu sprzężenia zwrotnego. Osiąga się to dostarczając testowanemu obiektowi (niestandardowymi metodami) wejściowych danych testowych lub zaślepek mogących odbierać dane wyjściowe z tego obiektu w miejsce rzeczywistych odbiorców.

Jarzma testowe stosuje się także w celu stworzenia rusztowania na poziomie oprogramowania pośredniego umożliwiającego jednoczesne testowanie środowiska języka programowania, systemu operacyjnego lub platformy sprzętowej.

Jarzma testowe możemy określać jako rusztowania do testów modułowych, gdyż stosuje się je przede wszystkim na poziomie testów modułowych. Ten typ narzędzi wspomaga wykonywanie testów modułowych równolegle z tworzeniem kodu.

#### Narzędzia do porównywania wyników (komparatory)

Komparatory testowe służą do porównywania plików, baz danych lub wyników testów. W skład narzędzi do wykonywania testów wchodzi zwykle komparatory dynamiczne, ale niekiedy porównanie wykonuje się dopiero po zakończeniu testowania przy pomocy odrębnego narzędzia do porównywania wyników. Komparator testowy posługuje się niekiedy wyrocznią, zwłaszcza zautomatyzowaną.

#### Narzędzia do pomiaru pokrycia (D)

Narzędzia do pomiaru pokrycia testowego są inwazyjne lub nieinwazyjne, zależnie od techniki wykonywania pomiaru, wykorzystywanej miary pokrycia oraz języka programowania.

Narzędzia do pomiaru pokrycia mierzą odsetek określonych konstrukcji w kodzie (np. instrukcji, rozgałęzień, decyzji, wywołań funkcji lub modułów), które zostały wykonane podczas testowania. Miary pokrycia wykazują, w jakim stopniu konstrukcje będące przedmiotem pomiaru są testowane przez dany zestaw testów.

### Narzędzia do testów zabezpieczeń

Narzędzia do testów zabezpieczeń (odporności) wyłapują wirusy komputerowe oraz identyfikują ataki mające na celu przeciążenie serwerów. Na przykład zapora ogniowa nie jest narzędziem testowym, ale może być stosowana do testowania zabezpieczeń. Inne narzędzia do testowania zabezpieczeń służą do poszukiwania określonych typów braku odporności i niezabezpieczonego dostępu do systemów.

## **6.1.6. Testowanie wydajności i monitorowanie wspierane narzędziami (K1)**

### Narzędzia do analizy dynamicznej (D)

Narzędzia do analizy dynamicznej odnajdują defekty dające się zaobserwować wyłącznie podczas wykonywania oprogramowania, takie jak zależności czasowe lub wycieki pamięci. Używa się ich zwykle podczas testowania modułowego, testowania integracji modułów oraz do testowania oprogramowania pośredniego.

### Narzędzia do testów wydajności, obciążenia i przeciążających

Narzędzia do testów wydajności monitorują i raportują działanie systemu w różnorodnych symulowanych warunkach użytkowania. Narzędzia te symulują obciążenie aplikacji, bazy danych albo komponentów środowiska systemu, takich jak sieć lub serwer. Nazwy tych narzędzi (na przykład narzędzia do testów obciążenia lub przeciążających) zwykle określają, do testowania jakiego aspektu wydajności są stosowane. Ich działanie polega zazwyczaj na wielokrotnym, automatycznym wykonywaniu sparametryzowanych testów.

### Narzędzia do monitorowania

Narzędzia do monitorowania systemów nie są ściśle rzecz biorąc narzędziami testowymi, ale dostarczają informacji niedostępnej w inny sposób, wykorzystywanej do celów testowania.

Narzędzia monitorujące na bieżąco analizują, weryfikują i raportują wykorzystanie określonych zasobów systemowych oraz ostrzegają o zagrożeniach systemu. Ponadto przechowują dane dotyczące wersji i build'ów oprogramowania, testaliów, a także umożliwiają śledzenie powiązań.

## **6.1.7. Narzędzia wspierające testowanie w określonych dziedzinach**

Niektóre narzędzia należące do opisywanych powyżej typów mogą być wyspecjalizowane do testowania specjalnego rodzaju oprogramowania. Na przykład istnieją specjalne narzędzia do testów wydajności aplikacji internetowych, narzędzia do analizy statycznej dedykowane poszczególnym platformom programistycznym lub narzędzia do analizy dynamicznej wykonywanej przede wszystkim pod kątem zabezpieczeń systemu.

Dostępne komercyjne zestawy narzędzi testowych bywają przeznaczone dla specjalnych zastosowań, na przykład dla systemów wbudowanych.

### **6.1.8. Wykorzystanie innych narzędzi (K1)**

Wyliczone wcześniej rodzaje narzędzi testowych nie są jedynymi stosowanymi przez testerów. Wykorzystywane są również np. arkusze kalkulacyjne, narzędzia SQL, debugery i narzędzia do zarządzania zasobami.

## 6.2. Skuteczne zastosowanie narzędzi: możliwe korzyści i zagrożenia (K2), 20 minut

### Terminologia

Testowanie w oparciu o dane, testowanie oparte na słowach kluczowych, język skryptowy.

#### 6.2.1. Możliwe korzyści i zagrożenia wynikające ze stosowania narzędzi wspierających testowanie (dla wszystkich rodzajów narzędzi) (K2)

Samo tylko zakupienie czy wypożyczenie narzędzia nie gwarantuje sukcesu. Każdy typ narzędzi wymaga zwykle dodatkowego wysiłku, aby osiągnąć rzeczywiste i trwałe korzyści z jego zastosowania. Istnieją zarówno potencjalne korzyści, jak i ryzyko związane z posługiwaniem się narzędziami do testowania.

Możliwe korzyści z wykorzystania narzędzi:

- Ograniczenie wielokrotnego wykonywania tej samej pracy (na przykład wykonywania testów regresyjnych, wielokrotnego wprowadzania tych samych danych testowych, lub kontroli zgodności ze standardami programowania).
- Większa jednolitość i powtarzalność (na przykład testy wykonywane przy pomocy narzędzia lub przypadki testowe uzyskane z wymagań).
- Obiektywna ocena (na przykład miary statyczne, pokrycie i zachowanie systemu).
- Łatwy dostęp do danych o testach lub testowaniu (na przykład statystyki i grafy obrazujące postęp testowania, liczba zdarzeń lub wydajność).

Zagrożenia wiążące się z zastosowaniem narzędzi:

- Nierealistyczne oczekiwania wobec narzędzia (zarówno jego funkcjonalności, jak i łatwości posługiwania się narzędziem).
- Zaniżone oszacowanie czasu, kosztu oraz wysiłku pierwszego wdrożenia narzędzia (w tym kosztów szkoleń oraz zewnętrznego wsparcia i doradztwa).
- Zaniżone oszacowanie czasu i wysiłku niezbędnego, aby uzyskać znaczące i stałe korzyści z zastosowania narzędzia (w tym konieczności zmian w procesie testowania oraz zapewnienia stałego doskonalenia sposobu wykorzystywania narzędzia).
- Zaniżone oszacowanie wysiłku związanego z utrzymaniem testaliów wygenerowanych przez narzędzie.
- Zbytnie uzależnienie od narzędzia (np. automatyczne wykonywanie testów zamiast projektowania testów lub testy automatyczne tam, gdzie ręczne są skuteczniejsze).

#### 6.2.2. Zagadnienia specjalne dla niektórych rodzajów narzędzi (K1)

##### Narzędzia wspierające wykonanie testów

Narzędzia wspierające wykonanie testów są sterowane przechowywanymi w formie elektronicznej skryptami, zaprojektowanymi tak, aby implementować wykonanie testów.

Osiągnięcie znaczących korzyści przy wykorzystaniu tego typu narzędzi zwykle wymaga znaczących nakładów.

Rejestrowanie skryptów testowych poprzez nagrywanie czynności wykonywanych podczas ręcznego testowania może się wydawać korzystne, ale ta metoda nie nadaje się do stosowania przy dużej liczbie zautomatyzowanych testów. Zarejestrowany skrypt jest liniowym odwzorowaniem, wykonanych podczas testu manualnego, czynności, stanowiących – wraz z danymi – elementy każdego skryptu. Ten rodzaj skryptu może okazać się zawodny w razie pojawienia się nieoczekiwanych wydarzeń.

Metodyka testowania w oparciu o dane oddziela testowe dane wejściowe (dane testowe), zwykle przechowywane w arkuszu kalkulacyjnym, oraz posługuje się ogólniejszym skrypcem testowym, odczytującym dane testowe i wykonującym ten sam test z różnymi danymi. Testerzy nieznający języka skryptowego mogą wówczas projektować nowe testy wprowadzając dane testowe wykorzystywane przez z góry zdefiniowane skrypty.

W metodyce testowania w oparciu o słowa kluczowe umieszcza się w arkuszu kalkulacyjnym zarówno słowa klucze określające, jaką czynność należy wykonać (nazywane też słowami czynnościowymi), jak i dane testowe. Testerzy, nawet nieznający języka skryptowego, mogą projektować testy posługując się słowami kluczowymi, które powinny być dostosowane do rodzaju testowanej aplikacji.

Każda z metodyk wymaga dobrej znajomości języka skryptowego, albo przez testerów, albo przez specjalistów od automatyzacji.

Niezależnie od stosowanej metodyki, niezbędne jest gromadzenie i przechowywanie wyników oczekiwanych testów, w celu ich wykorzystania do porównania z przyszłymi wynikami rzeczywistymi.

## Narzędzia do testów wydajności

Zastosowanie narzędzi do testów wydajności wymaga wsparcia ze strony eksperta w dziedzinie testów wydajnościowych przy projektowaniu testów oraz interpretacji ich wyników.

## Narzędzia do analizy statycznej

Narzędzia do analizy statycznej zastosowane do kodu źródłowego umożliwiają narzucanie standardów kodowania, ale generują bardzo wiele komunikatów, gdy używa się ich do już istniejącego kodu. Komunikaty ostrzeżeń nie blokują przetłumaczenia kodu źródłowego na wykonywalny, ale powinno się ich unikać, aby ułatwić przyszłe utrzymanie kodu. Skuteczne podejście polega na stopniowym wdrożeniu z zastosowaniem wstępnych filtrów, które blokują niektóre zbędne komunikaty.

## Narzędzia do zarządzania testami

Chcąc selekcjonować i przedstawiać informację w formacie jak najlepiej pasującym do bieżących potrzeb organizacji, narzędzia do zarządzania testowaniem muszą porozumiewać się z innymi narzędziami lub arkuszami kalkulacyjnymi. Raporty, aby były opłacalne, muszą być odpowiednio zaprojektowane oraz monitorowane i analizowane.

### 6.3. Wdrożenie narzędzia w organizacji (K1), 15 minut

#### Terminologia

Brak szczególnej terminologii w tym rozdziale.

#### Wprowadzenie

Główne zasady wdrożenia narzędzi w organizacji obejmują:

- Ocena dojrzałości organizacyjnej, silnych i słabych stron procesu, oraz identyfikację możliwości udoskonalenia procesu testowego wspomaganego narzędziami.
- Ocena wdrożenia wobec jasnych wymagań przy pomocy obiektywnych kryteriów.
- Wstępną próbę wdrożenia w celu sprawdzenia wymaganej funkcjonalności i zgodności narzędzia z celami.
- Ocena dostawcy narzędzia (w tym szkoleń, wsparcia dla klientów oraz aspektów ekonomicznych).
- Oszacowanie wewnętrznych potrzeb w zakresie doradztwa indywidualnego oraz wsparcia w posługiwaniu się narzędziem.

Wstępną próbę wdrożenia najlepiej przeprowadzić w formie niewielkiego projektu pilotażowego, co pozwoli na zminimalizowanie kosztów poniesionych w razie natknięcia się na znaczne trudności i niepowodzenia projektu pilotażowego.

Cele projektu pilotażowego:

- Szczegółowe zapoznanie się z narzędziem.
- Sprawdzenie, jak posługiwanie się narzędziem pasuje do istniejących procesów i praktyk, oraz identyfikacja ewentualnie potrzebnych zmian.
- Określenie procedur posługiwania się, utrzymywania, przechowywania narzędzia i związanych z nim testaliów (np. określenie zasad nazywania plików i testów, tworzenia bibliotek oraz określania szczegółowości scenariuszy testowych).
- Oszacowanie, na ile korzyści z zastosowania narzędzia dadzą się osiągnąć przy opłacalnych nakładach.

Czynniki, od których zależy powodzenie wdrożenia narzędzia w organizacji:

- Stopniowe (przyrostowe) wdrożenie narzędzia w pozostałej części organizacji, nie objętej projektem pilotażowym.
- Dostosowanie i udoskonalenie procesów tak, aby współdziałały z użyciem narzędzia.
- Udostępnienie szkoleń oraz doradztwa dla nowych użytkowników.
- Określenie wytycznych dla posługiwania się narzędziem.
- Wdrożenie metodyki gromadzenia i wykorzystywania doświadczeń z posługiwania się narzędziem.
- Nadzorowanie i monitorowanie wykorzystania oraz korzyści z zastosowania narzędzia.

## Referencje

6.2.2 Buwalda, 2001, Fewster, 1999

6.3 Fewster, 1999

## 7. Referencje

### **Normy**

ISTQB Glossary of terms used in Software Testing Version 1.0

[CMMI] Chrissis, M.B., Konrad, M. and Shrum, S. (2004) *CMMI, Guidelines for Process Integration and Product Improvement*, Addison Wesley: Reading, MA; por. rozdz. 2.1

[IEEE 829] IEEE Std 829™ (1998/2005) IEEE Standard for Software Test Documentation (obecnie w trakcie modyfikacji); por. rozdz. 2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6

[IEEE 1028] IEEE Std 1028™ (1997) IEEE Standard for Software Reviews; zob. rozdz.3.2

[IEEE 12207] IEEE 12207/ISO/IEC 12207-1996, Software life cycle processes; zob. rozdz. 2.1

[ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality; zob. rozdz. 2.3

### **Książki**

[Beizer, 1990] Beizer, B. (1990) *Software Testing Techniques* (2nd edition), Van Nostrand Reinhold: Boston zob. rozdz. 1.2, 1.3, 2.3, 4.2, 4.3, 4.4, 4.6

[Black, 2001] Black, R. (2001) *Managing the Testing Process* (2nd edition), John Wiley & Sons: New York; zob. rozdz. 1.1, 1.2, 1.4, 1.5, 2.3, 2.4, 5.1, 5.2, 5.3, 5.5, 5.6

[Buwalda, 2001] Buwalda, H. et al. (2001) *Integrated Test Design and Automation*, Addison Wesley: Reading, MA; zob. rozdz. 6.2

[Copeland, 2004] Copeland, L. (2004) *A Practitioner's Guide to Software Test Design*, Artech House: Norwood, MA; zob. rozdz. 2.2, 2.3, 4.2, 4.3, 4.4, 4.6

[Craig, 2002] Craig, Rick D. and Jaskiel, Stefan P. (2002) *Systematic Software Testing*, Artech House: Norwood, MA; zob. rozdz. 1.4.5, 2.1.3, 2.4, 4.1, 5.2.5, 5.3, 5.4

[Fewster, 1999] Fewster, M. and Graham, D. (1999) *Software Test Automation*, Addison Wesley: Reading, MA; zob. rozdz. 6.2, 6.3

[Gilb, 1993]: Gilb, Tom and Graham, Dorothy (1993) *Software Inspection*, Addison Wesley: Reading, MA; zob. rozdz. 3.2.2, 3.2.4

[Hetzel, 1988] Hetzel, W. (1988) *Complete Guide to Software Testing*, QED: Wellesley, MA; zob. rozdz. 1.3, 1.4, 1.5, 2.1, 2.2, 2.3, 2.4, 4.1, 5.1, 5.3

[Kaner, 2002] Kaner, C., Bach, J. and Pettitcord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons; zob. rozdz. 1.1, 4.5, 5.2

[Myers 1979] Myers, Glenford J. (1979) *The Art of Software Testing*, John Wiley & Sons; zob. rozdz. 1.2, 1.3, 2.2, 4.3

[van Veenendaal, 2004] van Veenendaal, E. (ed.) (2004) *The Testing Practitioner* (Chapters 6, 8, 10), UTN Publishers: The Netherlands; zob. rozdz. 3.2, 3.3

## **Załącznik A – Pochodzenie planu**

### ***Historia tego dokumentu***

Dokument został przygotowany przez roboczą grupę składającą się z członków wyłonionych przez International Software Testing Qualifications Board (ISTQB). Początkowo był przeglądany przez wybrany panel przeglądowy a potem przez reprezentantów wybranych z międzynarodowej społeczności zajmującej się testowaniem oprogramowania. Zasady zastosowane przy tworzeniu tego dokumentu pokazane są w Załączniku C.

Dokument ten jest planem dla Międzynarodowego Certyfikatu Podstawowego w Testowaniu Oprogramowania, pierwszego poziomu międzynarodowej kwalifikacji zaaprobowanego przez ISTQB ([www.istqb.org](http://www.istqb.org)). W czasie pisania tego dokumentu (2005) w skład ISTQB wchodziły Austria, Dania, Finlandia, Francja, Niemcy, Indie, Izrael, Japonia, Korea, Norwegia, Polska, Portugalia, Hiszpania, Szwecja, Szwajcaria, Holandia, Wielka Brytania i USA.

### ***Cele Certyfikatu Podstawowego***

Poznanie testowania jako podstawowej i profesjonalnej specjalizacji inżynierii oprogramowania.

Zapewnienie standardu dla rozwoju kariery testera.

Umożliwienie rozpoznawania profesjonalnie wykwalifikowanych testerów przez pracodawców, klientów i kolegów oraz podwyższenie statusu testerów.

Promowanie konsekwentnych i dobrych praktyk testowania w dyscyplinach inżynierii oprogramowania.

Zidentyfikowanie tematów testowania, które są istotne i wartościowe dla przemysłu.

Umożliwienie dostawcom oprogramowania wynajęcie certyfikowanych testerów i dzięki temu uzyskanie handlowej przewagi nad ich konkurentami przez zareklamowanie własnej polityki zatrudnienia testerów.

Zapewnienie testerom i osobom zainteresowanym testowaniem okazji nabycia rozpoznawanych na arenie międzynarodowej kwalifikacji w tym temacie.

### ***Cele międzynarodowej kwalifikacji (przyjęte na spotkaniu ISTQB w Sollentuna, listopad 2001)***

Umożliwienie porównania umiejętności testowania w różnych krajach.

Umożliwienie testerom łatwiejszego przekraczania granic krajów.

Umożliwienie wielonarodowym/międzynarodowym projektom wspólnego zrozumienia zagadnień z zakresu testowania.

Zwiększenie liczby wykwalifikowanych testerów na świecie.

Posiadanie większego wpływu/wartości jako inicjatywa umocowana międzynarodowo niż specyficzne podejście z jednego kraju.

Rozwinięcie wspólnej międzynarodowej grupy zrozumienia i wiedzy o testowaniu dzięki planowi i terminologii oraz wzrost poziomu wiedzy na temat testowania u wszystkich uczestników.

Promowanie testowania jako zawodu w wielu krajach.

Umożliwienie testerom uzyskania rozpoznawanych kwalifikacji w ich ojczystych językach.

Umożliwienie dzielenia się wiedzą i zasobami pomiędzy krajami.

Zapewnienie międzynarodowego poznania testerów i kwalifikacji z powodu uczestnictwa z wielu krajów.

## **Wymagania wobec kandydatów**

Podstawowym wymaganiem wobec kandydatów na Certyfikat ISTQB z Testowania Oprogramowania jest zainteresowanie testowaniem oprogramowania. Ponadto obowiązują następujące zalecenia:

- Kandydaci powinni mieć co najmniej sześciomiesięczne doświadczenie w wytwarzaniu oprogramowania albo w testach akceptacyjnych lub systemowych.
- Kandydaci powinni wziąć udział w kursie ISTQB (akredytowanym przez narodową organizację uznaną przez ISTQB).

## **Okoliczności powstania i historia Podstawowego Certyfikatu w Testowaniu Oprogramowania**

Niezależna certyfikacja testerów oprogramowania rozpoczęła się w Wielkiej Brytanii wraz z powstaniem w 1998 *Software Testing Board* ([www.bcs.org.uk/iseb](http://www.bcs.org.uk/iseb)) pod auspicjami ISEB (*the British Computer Society's Information Systems Examination Board*). W 2002 roku, niemiecka ASQF stworzyła niemiecki system kwalifikacji testerów ([www.asqf.de](http://www.asqf.de)). Niniejszy plan sporządzono na podstawie planów ISEB i ASQF. Jego treść jest częściowo nowa, częściowo zreorganizowana i zmieniona, zaś nacisk położony jest na zapewnienie testerom maksymalnie praktycznego wsparcia.

Istniejące Certyfikaty Podstawowe Testowania Oprogramowania (np. z ISEB, ASQF lub narodowej organizacji ISTQB), przyznane przed powstaniem certyfikatu międzynarodowego, będą uznawane za równoważne certyfikatowi międzynarodowemu. Certyfikat Podstawowy nie wygasa i nie potrzebuje być odnawiany. Data przyznania znajduje się na Certyfikacie.

Narodowe organizacje ISTQB nadzorują miejscowe zagadnienia w swoich krajach. Obowiązki narodowych organizacji określa ISTQB, lecz ich realizacja pozostawiona jest samym organizacjom. Do obowiązków organizacji krajowych należy akredytacja dostawców szkoleń i wyznaczenie egzaminów.

## Załącznik B – Cel nauki i poziomy wiedzy

Niniejszy plan określa opisane poniżej cele nauczania. Każdy z tematów planu uwzględniony jest w trakcie egzaminu zgodnie z określonym dla niego celem.

### Poziom 1: Zapamiętać (K1)

Kandydat rozpoznaje, pamięta i umie określić termin lub pojęcie.

#### **Przykład**

Rozpoznanie definicji „awarii”: jako:

- „brak funkcjonalności użytkownika końcowego lub innego interesariusza” lub jako
- „niezgodność rzeczywistego działania modułu lub systemu z działaniem lub wynikiem oczekiwanym”.

### Poziom 2: Zrozumieć (K2)

Kandydat potrafi uzasadnić lub wyjaśnić zagadnienia z zakresu tematu oraz podsumować, porównać, klasyfikować i podawać przykłady dla różnych pojęć z zakresu testowania.

#### **Przykłady**

Uzasadnić, dlaczego testy powinny być projektowane jak najwcześniej:

- W celu znalezienia błędów wówczas, gdy ich usunięcie jest mniej kosztowne.
- Aby najważniejsze błędy znaleźć wcześniej.

Wyjaśnić podobieństwa i różnice pomiędzy testowaniem integracyjnym i systemowym:

- Podobieństwa: testowanie więcej niż jednego modułu oraz możliwość testowania właściwości.
- Różnice: testowanie integracyjne skupia się na interfejsach oraz na interakcji między modułami, zaś testowanie systemowe skupia się na aspektach ogólnosystemowych, takich jak przetwarzanie obejmujące całościową funkcjonalność.

### Poziom 3: Zastosowanie (K3)

Kandydat potrafi wybrać prawidłowe zastosowanie pojęcia lub techniki i zastosować je do danego kontekstu.

#### **Przykład**

- Może zidentyfikować wartości graniczne dla dozwolonych i niedozwolonych klas równoważności.
- Może wybrać przypadki testowe z podanego diagramu przejść stanów, aby pokryć wszystkie przejścia.

## Referencje

(Dla poznawczych poziomów celów nauki)

Anderson, L.W. i Krathwohl, D. R.(eds) (2001) A Taxonomy for Learning, Teaching, and Accessing: A Revision of Bloom’s Taxonomy of Educational Objectives, Allyn& Bacon:

## **Załącznik C – Zasady dotyczące Planu poziomu podstawowego ISTQB / SJSI**

Zasady podane tutaj zostały użyte w projekcie i przeglądzie tego planu. (Po każdej zasadzie pokazany jest dużymi literami stenograficzny skrót zasady)

### **Ogólne zasady**

- SG1. Plan powinien być zrozumiały i przyswajalny przez osoby z doświadczeniem w testowaniu od 0 do 6 miesięcy (lub więcej). (6 MIESIĘCY)
- SG2. Plan powinien być raczej praktyczny niż teoretyczny. (PRAKTYCZNY)
- SG3. Plan powinien być jasny i niedwuznaczny dla jego czytelników. (JASNY)
- SG4. Plan powinien być zrozumiały dla ludzi z różnych krajów i dać się łatwo przetłumaczyć na różne języki. (PRZETŁUMACZALNY)
- SG5. Plan powinien używać Amerykańskiego Angielskiego. (AMERYKAŃSKI ANGIELSKI)

### **Bieżąca treść**

- SC1. Plan powinien obejmować ostatnie koncepcje dotyczące testowania i powinien odzwierciedlać najlepszą bieżącą praktykę w testowaniu oprogramowania tam, gdzie to generalnie uzgodniono. Plan podlega przeglądowi co trzy do pięciu lat. (OSTATNI)
- SC2. Plan powinien pomniejszać wpływy zależne od czasu, takie jak bieżące uwarunkowania rynkowe, aby mieć czas życia od trzech do pięciu lat. (AKTUALNY)

### **Cele nauki**

- LO1. Cele nauki powinny różnić się pomiędzy pozycjami do rozpoznania/zapamiętania (poznawczy poziom K1), pozycjami, które kandydat powinien zrozumieć koncepcyjnie (K2) i tymi, które kandydat powinien potrafić zastosować w praktyce (K3) (POZIOM WIEDZY)
- LO2. Opis treści powinien być spójny z celami nauki. (SPÓJNY Z CELAMI NAUKI)
- LO3. Aby zilustrować cele nauki, pytania próbnego egzaminu dla każdej większej sekcji powinny wynikać z planu. (CELE NAUKI-EGZAMIN)

### **Ogólna struktura**

- ST1. Struktura planu powinna być jasna i pozwolić na odsyłanie do i z innych części, z pytań egzaminacyjnych i z innych istotnych dokumentów. (ODSYŁANIE)
- ST2. Pokrywanie się pomiędzy sekcjami planu powinno być zminimalizowane. (POKRYWANIE SIĘ)
- ST3. Każda sekcja planu powinna mieć tą samą strukturę. (SPÓJNA STRUKTURA)
- ST4. Plan powinien zawierać wersję, datę wydania i numer strony na każdej stronie. (WERSJA)
- ST5. Plan powinien zawierać wskazówkę odnośnie potrzebnej ilości czasu dla każdego rozdziału (aby odzwierciedlić względną wagę każdego tematu). (POTRZEBNY CZAS)

### **Referencje**

- SR1. Źródła i referencje zostaną podane dla koncepcji w konspekcie, aby pomóc dostawcom szkoleń znaleźć więcej informacji na wybrany temat. (REFERENCJE)

SR2. Tam, gdzie nie ma zidentyfikowanych i jasnych źródeł powinno się dostarczyć więcej szczegółów w konspekcie. Na przykład, definicje są w słowniku, a więc w konspekcie przytoczono jedynie terminologie. (SZCZEGÓŁY BEZ REFERENCJI)

### Źródła informacji

Terminologia użyta w konspekcie, dla pojęć używanych w testowaniu oprogramowania, zdefiniowana jest w Słowniku ISTQB. Wersja słownika jest dostępna na stronie ISTQB lub SJSI.

Lista zalecanych książek na temat testowania oprogramowania podana jest w konspekcie. Główna lista książek jest częścią sekcji Referencje.

## **Załącznik D – Informacja dla dostawców szkoleń**

Dla każdego z głównych tematów planu przeznaczony jest określony w minutach czas jego trwania. Celem tego jest zarówno wyznaczenie względnych proporcji poszczególnych sekcji tematycznych akredytowanego szkolenia względem siebie, jak i wyznaczenie przybliżonego minimalnego czasu, który należy przeznaczyć na nauczanie poszczególnych tematów. Dostawcy szkoleń mogą poświęcić im więcej czasu, zaś kandydaci mogą spędzić dodatkowy czas na lekturze i własnych badaniach. Kolejność omawiania zagadnień podczas kursu nie musi być taka sama jak w konspekcie.

Plan zawiera odwołania do uznanych norm, z których należy skorzystać podczas przygotowywania materiałów szkoleniowych. Zalecane jest korzystanie z tej wersji normy, do której odwołuje się bieżąca wersja planu. Można odwoływać się lub wykorzystywać także inne publikacje, wzorce lub normy poza określonymi w konspekcie, ale nie będą one przedmiotem egzaminu.

Następujące obszary tematyczne planu wymagają ćwiczeń praktycznych:

### **4.3 Techniki na podstawie specyfikacji lub czarnoskrzynkowe**

Praktyczne, krótkie ćwiczenia powinny dotyczyć czterech technik: podziału na klasy równoważności, analizy warunków brzegowych, testowania tabeli decyzyjnej oraz testowanie przejść stanów. Wykład oraz ćwiczenia dotyczące tych technik powinny być zrobione na podstawie referencji podanych dla każdej z nich.

### **4.4 Techniki na podstawie struktury lub białoskrzynkowe**

Praktyczne, krótkie ćwiczenia powinny dotyczyć sprawdzenia, czy dany zestaw przypadków testowych pozwala na 100% pokrycia instrukcji lub 100% pokrycia decyzji, a także projektowania przypadków testowych dla określonych przepływów sterowania.

### **5.6 Zarządzanie incydentami**

Należy zastosować krótkie ćwiczenie, polegające albo na napisaniu, albo ocenie pisemnego zgłoszenia incydentu.

## Indeks

### A

analiza statyczna, 29, 30, 34  
analiza wartości brzegowych, 35, 39  
analiza wpływu, 28  
automatyzacja, 26, 47  
automatyzacja testów, 47  
awaria, 10, 11

### B

błąd, 10, 11, 16

### C

cecha, 37  
cele testowania, 15, 16, 20  
cykl życia, 21, 43  
czas odpowiedzi, 27  
częstotliwość awarii, 51  
czynności poza testowaniem, 55  
czynności testowe, 14  
czynność testowa, 21

### D

dane testowe, 15, 37, 62, 66  
dane wejściowe, 62, 66  
debager, 59  
debugowanie, 13, 27  
decyzja, 12  
defekt, 10, 11, 13, 31, 34  
diagram przejść pomiędzy stanami, 40  
dokumentacja testowa, 53  
domyślanie się błędów, 49  
działanie, 11, 40, 48, 63

### E

efekt próbnika, 59  
efektywność, 11, 33

### F

funkcja, 37, 62  
funkcjonalność, 24, 26, 54, 72

### H

harmonogram, 37  
harmonogram wykonania testu, 37

### I

incydent, 15, 56  
inspekcja, 29, 31, 32, 33  
instrukcja, 56  
integracja, 15, 23  
interfejs, 34, 59

iteracyjny, 23

### J

jakość, 10, 11, 12, 18, 46, 47, 54  
jarzmo testowe, 59, 62  
język skryptowy, 62, 65

### K

kierownik jakości, 46  
kierownik testów, 18, 46  
kod, 11, 12, 13, 18, 21, 23, 30, 34, 41, 61, 66  
kod źródłowy, 21  
kompilator, 34  
koszty testowania, 51  
kryteria wejścia, 48  
kryteria wejściowe, 31

### L

lista kontrolna, 32  
log testu, 15  
logowanie wyników testów, 60

### Ł

łatwość testowania, 48

### M

menedżer testów, 46  
metodyka testowa, 48, 51, 55, 66  
metodyka testowania, 55, 66  
miara, 27  
migracja, 20, 28  
model automatu skończonego (przejść stanów), 26  
model przepływu procesu, 26  
model wytwarzania, 21  
moduł, 24, 62  
modyfikacja, 20  
monitorowanie postępu testów, 51  
monitorowanie testowania, 51

### N

nadzorowanie procesu testowego, 44  
nadzór nad testowaniem, 15, 51  
najistotniejsze, 55  
narzędzia do analizy dynamicznej, 63  
narzędzia do analizy statycznej, 34, 61, 63, 66  
narzędzia do zarządzania testami, 59, 66  
narzędzia wspierające projektowanie testów, 61  
narzędzia wspierające testowanie, 63  
narzędzie, 34, 59, 65  
narzędzie do analizy statycznej, 59  
narzędzie do pomiaru pokrycia, 59  
narzędzie do testów wydajności, 59  
narzędzie do testów wydajnościowych, 59  
narzędzie do testów zabezpieczeń, 59  
narzędzie do zarządzania konfiguracją, 59

narzędzie do zarządzania wymaganiami, 59  
narzędzie wspierające wykonanie testu, 59  
niezależność testowania, 46  
niezależny tester, 46  
niezależny zespół testowy, 18, 24  
niezawodność, 11, 13, 26, 54, 59

## O

ocena, 13, 15, 16, 49, 51, 65  
oczekiwany wynik, 35  
oparte na słowach kluczowych, 65  
operacyjne testy akceptacyjne, 23, 25  
oprogramowanie, 11, 17, 18, 21, 25, 27, 38, 40, 42, 46, 54, 55  
oprogramowanie wbudowane, 40  
oprogramowanie z półki, 21, 25  
organizacja wytwarzająca oprogramowanie, 24  
osoba wykonująca przegląd, 56  
oszacowanie, 44, 47, 49, 65, 67

## P

plan testów, 15, 48  
planowanie integracji, 24  
planowanie ryzyka, 44  
planowanie testowania, 47, 48  
planowanie testów, 15, 24, 53  
pluskwa, 10, 11  
podejście, 21, 24, 27, 35, 37, 42, 66, 70  
podejście białoskrzynkowe, 35  
podejście czarnoskrzynkowe, 35  
podejście strukturalne, 27  
podstawa testów, 13, 15, 23  
pokrycie, 15, 26, 27, 35, 39, 41, 48, 51, 60, 65  
pokrycie danych wejściowych, 39  
pokrycie decyzji, 41  
pokrycie instrukcji kodu, 41  
pokrycie kodu, 26, 41, 48  
pokrycie testowe, 15, 27, 51  
pokrycie wymagań, 51  
polityka testów, 48  
pomyłka, 10, 11, 16  
ponownie przetestowane, 52  
poziom testów, 21  
pracochłonność testowania, 49  
procedura testowa, 35  
proces biznesowy, 41  
proces testowania, 13  
proces testowy, 10, 15, 29, 30  
projekt, 16, 17  
projektowanie przypadków testowych, 35, 37  
projektowanie testów, 16, 21, 22, 37, 49  
prototypowanie, 21  
przebieg wykonania testów, 51  
przegląd, 13, 29, 30, 31, 32, 33, 47  
przegląd formalny, 31  
przegląd koleżeński, 31, 32  
przegląd nieformalny, 29, 31, 32  
przegląd techniczny, 29, 31, 32  
przejrzanie, 29, 31, 32  
przepływ, 34, 41  
przepływ danych, 34  
przepływ sterowania, 34, 41  
przetestowanie właściwości, 26

przypadek testowy, 13, 35, 37  
przypadek użycia, 40

## R

raport testowy, 51  
raportowanie incydentów, 60  
raportowanie testów, 16, 51  
Rational Unified Process (RUP), 21  
rejestracja danych testowych, 62  
rozbudowa i udoskonalanie, 28  
rusztowanie testowe, 62  
ryzyko, 11, 12, 18, 24, 27, 36, 44, 50, 54, 55, 65  
ryzyko produktowe, 54  
ryzyko projektowe, 54  
ryzyko związane z produktem, 54, 55

## S

scenariusz testowy, 26, 27  
skrypt testowy, 37  
specyfikacja, 26, 28, 37  
specyfikacja procedury testowej, 37  
sprzęt, 23, 54  
staranność testowania, 24, 27  
statyczne techniki testowania, 29, 30  
sterownik, 59  
strategia testowa, 15, 48  
strategia testowania, 15  
system, 11, 13, 14, 21, 22, 23, 24, 26, 28, 39, 40, 71  
system operacyjny, 23  
szybkie wytwarzanie oprogramowania, 21

## Ś

śledzenie powiązań, 35, 37, 60, 63  
środowisko produkcyjne, 24  
środowisko testowe, 15, 16, 23, 51

## T

tablica decyzyjna, 24, 39  
techniki projektowania testów, 35, 36, 38  
techniki testowania., 55  
tester, 1, 18, 32, 46, 56  
testować, 12, 18, 24, 37, 55  
testowanie, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 34, 35, 39, 40, 41, 42, 44, 46, 47, 48, 49, 51, 54, 55, 58, 60, 61, 62, 63, 65, 72, 75  
testowanie akceptacyjne, 21, 22, 23, 25  
testowanie akceptacyjne przez użytkownika, 23, 25  
testowanie alfa, 23, 25  
testowanie beta, 23, 25  
testowanie białoskrzynkowe, 26, 41  
testowanie decyzyjne, 35, 41  
testowanie dynamiczne, 30, 34  
testowanie eksploracyjne, 42, 48, 49  
testowanie funkcji, 26  
testowanie funkcjonalne, 24, 26  
testowanie gruntowne, 14  
testowanie instrukcji, 35, 41  
testowanie integracyjne, 21, 22, 23, 72  
testowanie integracyjne modułów, 21  
testowanie integracyjne systemów, 21

testowanie modułowe, 13, 21, 23, 35  
testowanie na podstawie ryzyka, 49, 54, 55  
testowanie na podstawie struktury, 41  
testowanie нефункционалне, 26  
testowanie niezależne, 18  
testowanie niezawodności, 26  
testowanie obciążeniowe, 26  
testowanie odporności, 23  
testowanie potwierdzające, 13, 15, 16, 26, 27, 60  
testowanie przeciążające, 26  
testowanie przejść pomiędzy stanami, 39, 40  
testowanie regresywne, 15, 16, 26, 27  
testowanie statyczne wspierane narzędziami, 61  
testowanie stochastyczne, 49  
testowanie strukturalne, 23, 26, 27, 41  
testowanie systemowe, 21, 23, 24, 72  
testowanie użyteczności, 26  
testowanie w fazie utrzymania, 20, 28  
testowanie w oparciu o przypadki użycia, 39, 40  
testowanie w oparciu o specyfikację, 26  
testowanie właściwości, 24, 26  
testowanie współdziałania, 26  
testowanie wydajnościowe, 26  
testowanie zabezpieczeń, 26  
testowanie związane ze zmianami, 27  
testy, 13, 14, 15, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 35, 37, 39, 40, 42, 44, 49, 61, 65, 66, 72  
testy akceptacji produkcyjnej, 25  
testy akceptacyjne zgodności legislacyjnej, 23, 25  
testy akceptacyjne zgodności z umową, 23, 25  
testy integracji z infrastrukturą, 22  
testy integracyjne systemów, 24  
testy migracji danych, 28  
testy modułowe, 23  
testy modułu, 26  
testy odbiorcze i wdrożeniowe, 22  
testy regresywne, 21, 28, 37  
testy w środowisku produkcyjnym, 23, 25  
tworzenie specyfikacji testów wspierane narzędziami, 61  
typowe metryki, 44  
typy testów, 20, 27

## U

usterka, 11

utrzymywalność, 11  
użyteczność, 11, 26, 54

## W

walidacja, 21  
warunek testowy, 15, 35, 37  
warunki wejściowe, 39  
warunki wyjścia, 15, 31  
wdrożenie, 15, 55, 58, 67  
wejściowe dane testowe, 61  
weryfikacja, 13, 16, 21, 23, 25, 26  
wstępujące, 24  
wycofanie systemu, 20, 28  
wydajność, 54, 65  
wykonywanie testów, 47, 62, 65  
wykonywanie testów wspierane narzędziami, 62  
wymaganie, 13  
wyniki testów, 48, 51  
wysoka wartość miary złożoności, 34  
wytwarzanie sterowane testowaniem, 23  
wytwarzanie w modelu iteracyjnym, 21

## Z

zabezpieczenia, 54  
zachowanie systemu, 65  
zadania testowe, 46, 47, 48  
zagrożenie, 50  
zakres testowania, 48  
zarządzanie incydentami, 45, 56, 75  
zarządzanie konfiguracją, 44, 53  
zarządzanie testowaniem, 44, 59  
zarządzanie testowaniem wspierane narzędziami, 59  
zarządzanie wersjami, 59  
zarządzaniu ryzykiem, 55  
zdolność do konserwacji, 30  
zestawy testów do testów regresywnych, 27  
zgadywanie błędów, 18, 42  
zgłoszenie, 17, 56  
zgłoszenie incydentu, 56  
złożoność, 34, 61  
zmiana, 52  
zmiana wag testów, 52