# Certyfikowany tester

# Sylabus poziomu podstawowego

wersja 4.0

(wersja PL 4.0.0.0)

International Software Testing Qualifications Board


# Informacja o prawach autorskich

# Historia zmian

| Wersja | Data | Uwagi |
|---|---|---|
| CTFL v4.0 | 21.04.2023 r. | CTFL v4.0 — wersja do publikacji |
| CTFL v3.1.1 | 01.07.2021 r. | CTFL v3.1.1 — aktualizacja informacji o prawach autorskich i logo |
| CTFL v3.1 | 11.11.2019 r. | CTFL v3.1 — wydanie pielęgnacyjne z niewielkimi aktualizacjami |
| ISTQB 2018 | 27.04.2018 r. | CTFL v3.0 — potencjalna wersja do publikacji |
| ISTQB 2011 | 01.04.2011 r. | Wydanie pielęgnacyjne sylabusa CTFL |
| ISTQB 2010 | 30.03.2010 r. | Wydanie pielęgnacyjne sylabusa CTFL |
| ISTQB 2007 | 01.05.2007 r. | Wydanie pielęgnacyjne sylabusa CTFL |
| ISTQB 2005 | 01.07.2005 r. | Certyfikowany tester — sylabus poziomu podstawowego, wersja 1.0 |
| ASQF v2.2 | lipiec 2003 r. | Sylabus ASQF — poziom podstawowy, wersja 2.2: „Lehrplan Grundlagen des Software-testens” |
| ISEB v2.0 | 25.02.1999 r. | ISEB — testowanie oprogramowania — sylabus poziomu podstawowego, wersja 2.0 |

# Historia zmian dla polskiej wersji sylabusa

| Wersja | Data | Uwagi |
|---|---|---|
| 4.0.0.0 | 07.07.2023 | Publikacja polskiego tłumaczenia sylabusa |
| 0.3 | 01.06.2023 | Przegląd i wprowadzanie zmian – Zespół SJSI |
| 0.2 | 21.05.2023 | Przegląd tłumaczenia – Zespół SJSI |
| | 12.05.2023 | Udostępnienie przez ISTQB® wersji końcowej |
| 0.1 | 28.04.2023 | Tłumaczenie wersji beta: KONTEKST A. Wolski sp. komandytowa |

# Spis treści

# Podziękowania

Niniejszy dokument został formalnie zatwierdzony do publikacji przez Zgromadzenie Ogólne ISTQB® w dniu 21 kwietnia 2023 r.

Dokument został opracowany przez zespół złożony z przedstawicieli połączonych grup roboczych ISTQB® ds. poziomu podstawowego i testowania zwinnego w składzie: Laura Albert, Renzo Cerquozzi (wiceprzewodniczący), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (współprzewodniczący), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (współprzewodniczący), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie Ulrich (wiceprzewodnicząca) i Eshraka Zakaria.

Członkowie zespołu pragną podziękować Stuartowi Reidowi, Patricii McQuaid i Leanne Howard za redakcję techniczną oraz zespołowi recenzentów i Radom Krajowym za sugestie i wskazówki.

W procesie przeglądu, zgłaszania uwag i głosowania nad niniejszym sylabusem uczestniczyły następujące osoby: Adam Roman, Adam Ścierski, Ágota Horváth, Ainsley Rood, Ale Rebon Portillo, Alessandro Collino, Alexander Alexandrov, Amanda Logue, Ana Ochoa, André Baumann, André Verschelling, Andreas Spillner, Anna Miazek, Armin Born, Arnd Pehl, Arne Becher, Attila Gyúri, Attila Kovács, Beata Karpińska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chinthaka Indikadahena, Chris Van Bael, Ciaran O'Leary, Claude Zhang, Cristina Sobrero, Dandan Zheng, Dani Almog, Daniel Säther, Daniel van der Zwan, Danilo Magli, Darvay Tamás Béla, Dawn Haynes, Dena Pauletti, Dénes Medzihradszky, Doris Dötzer, Dot Graham, Edward Weller, Erhardt Wunderlich, Eric Riou Du Cosquer, Florian Fieber, Fran O'Hara, François Vaillancourt, Frans Dijkman, Gabriele Haller, Gary Mogyorodi, Georg Sehl, Géza Bujdosó, Giancarlo Tomasig, Giorgio Pisani, Gustavo Márquez Sosa, Helmut Pichler, Hongbao Zhai, Horst Pohlmann, Ignacio Trejos, Ilia Kulakov, Ine Lutterman, Ingvar Nordström, Iosif Itkin, Jamie Mitchell, Jan Giesen, Jean-Francois Riverin, Joanna Kazun, Joanne Tremblay, Joëlle Genois, Johan Klintin, John Kurowski, Jörn Münzel, Judy McKay, Jürgen Beniermann, Karol Frühauf, Katalin Balla, Kevin Kooh, Klaudia Dussa-Zieger, Klaus Erlenbach, Klaus Olsen, Krisztián Miskó, Laura Albert, Liang Ren, Lijuan Wang, Lloyd Roden, Lucjan Stapp, Mahmoud Khalaili, Marek Majernik, Maria Clara Choucair, Mark Rutz, Markus Niehammer, Martin Klonk, Márton Siska, Matthew Gregg, Matthias Hamburg, Mattijs Kemmink, Maud Schlich, May Abu-Sbeit, Meile Posthuma, Mette Bruhn-Pedersen, Michal Tal, Michel Boies, Mike Smith, Miroslav Renda, Mohsen Ekssir, Monika Stocklein Olsen, Murian Song, Nicola De Rosa, Nikita Kalyani, Nishan Portoyan, Nitzan Goldenberg, Ole Chr. Hansen, Patricia McQuaid, Patricia Osorio, Paul Weymouth, Paweł Kwasik, Peter Zimmerer, Petr Neugebauer, Piet de Roo, Radosław Smilgin, Ralf Bongard, Ralf Reissing, Randall Rice, Rik Marselis, Rogier Ammerlaan, Sabine Gschwandtner, Sabine Uhde, Salinda Wickramasinghe, Salvatore Reale, Sammy Kolluru, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Surabhi Bellani, Szilard Szell, Tamás Gergely, Tamás Horváth, Tatiana Sergeeva, Tauhida Parveen, Thaer Mustafa, Thomas Eisbrenner, Thomas Harms, Thomas Heller, Tobias Letzkus, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo i Zsolt Hargitai.

Grupa robocza ds. poziomu podstawowego ISTQB® (wydanie 2018): Klaus Olsen (przewodniczący), Tauhida Parveen (wiceprzewodnicząca), Rex Black (kierownik projektu), Eshraka Zakaria, Debra Friedenberg, Ebbe Munk, Hans Schaefer, Judy McKay, Marie Walsh, Meile Posthuma, Mike Smith, Radosław Smilgin, Stephanie Ulrich, Steve Toms, Corne Kruger, Dani Almog, Eric Riou du Cosquer, Igal Levi, Johan Klintin, Kenji Onishi, Rashed Karim, Stevan Zivanovic, Sunny Kwon, Thomas Müller, Vipul Kocher i Yaron Tsubery. Członkowie zespołu składają podziękowania powyższym osobom oraz wszystkim Radom Krajowym za sugestie.

Grupa robocza ds. poziomu podstawowego ISTQB® (wydanie 2011): Thomas Müller (przewodniczący), Debra Friedenberg. Członkowie zespołu podstawowego składają podziękowania zespołowi recenzentów (w składzie: Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier, Hans Schaefer, Stephanie Ulrich i Erik van Veenendaal) oraz wszystkim Radom Krajowym za sugestie dotyczące bieżącej wersji sylabusa.

Grupa robocza ds. poziomu podstawowego ISTQB® (wydanie 2010): Thomas Müller (przewodniczący), Rahul Verma, Martin Klonk i Armin Beer. Członkowie zespołu podstawowego składają podziękowania zespołowi recenzentów (w składzie: Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams i Erik van Veenendaal) oraz wszystkim Radom Krajowym za sugestie.

Grupa robocza ds. poziomu podstawowego ISTQB® (wydanie 2007): Thomas Müller (przewodniczący), Dorothy Graham, Debra Friedenberg i Erik van Veenendaal. Członkowie zespołu podstawowego składają podziękowania zespołowi recenzentów (w składzie: Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson i Wonil Kwon) oraz wszystkim Radom Krajowym za sugestie.

Grupa robocza ds. poziomu podstawowego ISTQB® (wydanie 2005): Thomas Müller (przewodniczący), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson i Erik van Veenendaal. Członkowie zespołu podstawowego składają podziękowania zespołowi recenzentów oraz wszystkim Radom Krajowym za sugestie.

# 0. Wstęp

## 0.1. Cel niniejszego sylabusa

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

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

## 0.2. Certyfikowany tester — poziom podstawowy w testowaniu oprogramowania

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

## 0.3. Ścieżki kariery dla testerów

System stworzony przez ISTQB® umożliwia osobom zajmującym się testowaniem dostęp do szerokiej i szczegółowej wiedzy przydatnej na każdym etapie kariery. Posiadacze certyfikatu ISTQB® na poziomie podstawowym mogą być również zainteresowani certyfikatami w obszarze Core na poziomie zaawansowanym (analityk testów, techniczny analityk testów, kierownik testów), a następnie na poziomie eksperckim (zarządzanie testami, doskonalenie procesu testowego), natomiast osoby chcące rozwijać swoje umiejętności w dziedzinie praktyk testowania w środowiskach zwinnych powinny wziąć pod uwagę certyfikat technicznego testera zwinnego lub certyfikat ATLaS (Agile Test Leadership at Scale). Z kolei

ścieżka Specialist umożliwia ścisłą specjalizację w dziedzinach, w których stosowane są wyspecjalizowane podejścia do testowania i czynności testowe (np. automatyzacja testów, testowanie z wykorzystaniem sztucznej inteligencji, testowanie oparte na modelu, testowanie aplikacji mobilnych), które są związane z konkretnymi obszarami testowania (np. testowanie wydajnościowe, testowanie użyteczności, testowanie akceptacyjne, testowanie zabezpieczeń), bądź w których skupia się specjalistyczna wiedza w dziedzinie testowania dotycząca określonych sektorów przemysłu (takich jak branża automotive czy gry). Najnowsze informacje na temat systemu certyfikacji testerów ISTQB® można uzyskać w serwisie www.istqb.org.

## 0.4. Cele biznesowe

W tym podrozdziale wymieniono 14 celów biznesowych, które powinna realizować osoba otrzymująca certyfikat na poziomie podstawowym.

Certyfikowany tester na poziomie podstawowym potrafi realizować następujące cele:

| | |
|---|---|
| FL-BO1 | Znajomość istoty testowania i wynikających z niego korzyści |
| FL-BO2 | Znajomość podstawowych pojęć związanych z testowaniem oprogramowania |
| FL-BO3 | Identyfikowanie podejścia do testowania i czynności testowych, które mają być realizowane w zależności od kontekstu testowania |
| FL-BO4 | Dokonywanie oceny i podnoszenie jakości dokumentacji |
| FL-BO5 | Zwiększanie skuteczności i efektywności testowania |
| FL-BO6 | Dopasowywanie procesu testowego do cyklu wytwarzania oprogramowania |
| FL-BO7 | Znajomość zasad zarządzania testami |
| FL-BO8 | Sporządzanie i udostępnianie przejrzystych, zrozumiałych raportów o defektach |
| FL-BO9 | Znajomość czynników wpływających na priorytety i pracochłonność testowania |
| FL-BO10 | Praca w zespole interdyscyplinarnym |
| FL-BO11 | Znajomość ryzyk i korzyści związanych z automatyzacją testów |
| FL-BO12 | Identyfikowanie niezbędnych umiejętności wymaganych w związku z testowaniem |
| FL-BO13 | Znajomość wpływu ryzyka na testowanie |
| FL-BO14 | Sprawne raportowanie na temat postępu i jakości testów |

## 0.5. Cele nauczania objęte egzaminem i poziomy wiedzy

Cele nauczania wspomagają osiąganie celów biznesowych i stanowią wytyczne do tworzenia egzaminów Certyfikowany tester poziom podstawowy. Poziomy wiedzy dla poszczególnych celów nauczania przedstawiono na początku każdego rozdziału.

Poziomy te sklasyfikowano następująco:

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

Dalsze szczegóły i przykłady celów nauczania podano w Załączniku A. Definicje wszystkich pojęć wymienionych jako słowa kluczowe pod tytułami rozdziałów należy zapamiętać (K1), nawet jeśli nie wspomniano o tym wyraźnie w celach nauczania.

## 0.6. Egzamin certyfikacyjny na poziomie podstawowym

Egzamin umożliwiający uzyskanie certyfikatu na poziomie podstawowym jest oparty na niniejszym sylabusie. Przy udzielaniu odpowiedzi na pytania egzaminacyjne może być konieczne skorzystanie z materiału obejmującego więcej niż jeden rozdział tego sylabusa. Przedmiotem egzaminu może być treść wszystkich części sylabusa z wyjątkiem wstępu i załączników. W dokumencie znajdują się również odwołania do norm/standardów i książek (rozdział 7), ale ich treść nie może być przedmiotem egzaminu w zakresie wykraczającym poza informacje streszczone w tym sylabusie. Więcej informacji na ten temat zawiera dokument dotyczący struktury i zasad przeprowadzania egzaminów na poziomie podstawowym (*Foundation Level Examination Structures and Rules*).

## 0.7. Akredytacja

Rada Krajowa ISTQB® może dokonywać akredytacji dostawców szkoleń, którzy oferują materiały dydaktyczne zgodne z niniejszym sylabusem. Wytyczne dotyczące akredytacji należy uzyskać od Rady Krajowej lub organu dokonującego akredytacji. Akredytowane szkolenie jest uznawane za zgodne z niniejszym sylabusem i może obejmować egzamin ISTQB®. Wytyczne dotyczące akredytacji w zakresie niniejszego sylabusa są zgodne z ogólnymi wytycznymi dotyczącymi akredytacji opublikowanymi przez grupę roboczą ds. zarządzania procesami i zgodności.

## 0.8. Odniesienia do norm i standardów

Sylabus poziomu podstawowego zawiera odniesienia do norm i standardów (np. IEEE lub ISO). Celem tych odniesień jest stworzenie ram pojęciowych (tak jak w przypadku odniesień do normy ISO 25010 w zakresie charakterystyk jakościowych) lub odesłanie czytelnika do źródła, z którego może skorzystać w celu uzyskania dodatkowych informacji. Należy jednak zaznaczyć, że treść norm i standardów nie jest przedmiotem egzaminu. Więcej informacji na temat norm i standardów zawiera rozdział 7.

## 0.9. Ciągła aktualizacja

W branży oprogramowania zachodzą dynamiczne zmiany. Aby uwzględnić zmieniającą się sytuację i zapewnić interesariuszom dostęp do przydatnych, aktualnych informacji, grupy robocze ISTQB® stworzyły listę odsyłaczy do dokumentów pomocniczych i zmian w normach/standardach, która jest dostępna w witrynie www.istqb.org. Powyższe informacje nie są przedmiotem egzaminu dotyczącego sylabusa poziomu podstawowego.

## 0.10. Poziom szczegółowości

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

- ogólne cele dydaktyczne opisujące założenia poziomu podstawowego;
- wykaz terminów (słów kluczowych), które muszą zapamiętać uczestnicy szkolenia;
- cele nauczania w poszczególnych obszarach wiedzy, opisujące efekty kształcenia o charakterze poznawczym;
- opis najważniejszych pojęć, w tym odsyłacze do uznanych źródeł.

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

## 0.11. Struktura sylabusa

Dokument podzielono na sześć rozdziałów zawierających treści będące przedmiotem egzaminu. Nagłówek najwyższego poziomu zawiera informację o czasie trwania szkolenia z zakresu danego rozdziału (nie podano czasu trwania podrozdziałów i mniejszych jednostek redakcyjnych). W przypadku akredytowanych szkoleń na przekazanie wiedzy zawartej w sylabusie potrzeba co najmniej 1135 min (18 godz. 55 min) wykładu. Czas ten podzielono na poszczególne rozdziały w następujący sposób:

- Rozdział 1. Podstawy testowania — 180 minut
    - Kandydat poznaje podstawowe zasady związane z testowaniem, powody, dla których testowanie jest niezbędne, oraz cele testów.
    - Kandydat poznaje proces testowy oraz najważniejsze czynności testowe i testalia.
    - Kandydat dowiaduje się, jakie umiejętności są niezbędne w testowaniu.
- Rozdział 2. Testowanie w cyklu wytwarzania oprogramowania — 130 minut
    - Kandydat dowiaduje się, w jaki sposób testowanie jest uwzględniane w różnych podejściach do wytwarzania oprogramowania.
    - Kandydat poznaje pojęcia związane z podejściem „najpierw test” i metodyką DevOps.
    - Kandydat uzyskuje wiedzę na temat poszczególnych poziomów testów i typów testów oraz testowania pielęgnacyjnego.
- Rozdział 3. Testowanie statyczne — 80 minut
    - Kandydat poznaje podstawy testowania statycznego oraz proces uzyskiwania informacji zwrotnych i przeprowadzania przeglądu.

- Rozdział 4. Analiza i projektowanie testów — 390 minut
    - Kandydat dowiaduje się, jak należy stosować techniki czarnoskrzynkowe, białoskrzynkowe i oparte na doświadczeniu, aby tworzyć przypadki testowe na podstawie różnych produktów pracy związanych z oprogramowaniem.
    - Kandydat poznaje podejście do testowania oparte na współpracy.
- Rozdział 5. Zarządzanie czynnościami testowymi — 335 minut
    - Kandydat poznaje ogólne zasady planowania testów i szacowania ich pracochłonności.
    - Kandydat dowiaduje się, jak ryzyka mogą wpływać na zakres testowania.
    - Kandydat dowiaduje się, jak należy monitorować i nadzorować czynności testowe.
    - Kandydat dowiaduje się, w jaki sposób zarządzanie konfiguracją wspomaga testowanie.
    - Kandydat uczy się zgłaszania defektów w przejrzysty i zrozumiały sposób.
- Rozdział 6. Narzędzia testowe — 20 minut
    - Kandydat uczy się klasyfikować narzędzia oraz poznaje ryzyka i korzyści wynikające z automatyzacji testów.

# 1. Podstawy testowania — 180 minut

**Słowa kluczowe**

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

**Cele nauczania w rozdziale 1:**

### 1.1 Co to jest testowanie?

FL-1.1.1 (K1) Kandydat wskazuje typowe cele testów.

FL-1.1.2 (K2) Kandydat odróżnia testowanie od debugowania.

### 1.2 Dlaczego testowanie jest niezbędne?

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

FL-1.2.2 (K1) Kandydat pamięta, jaka jest relacja między testowaniem a zapewnieniem jakości.

FL-1.2.3 (K2) Kandydat odróżnia podstawową przyczynę, pomyłkę, defekt i awarię.

### 1.3 Zasady testowania

FL-1.3.1 (K2) Kandydat objaśnia siedem zasad testowania.

### 1.4 Czynności testowe, testalia i role związane z testami

FL-1.4.1 (K2) Kandydat podsumowuje poszczególne czynności i zadania testowe.

FL-1.4.2 (K2) Kandydat wyjaśnia wpływ kontekstu na proces testowy.

FL-1.4.3 (K2) Kandydat rozróżnia testalia wspomagające czynności testowe.

FL-1.4.4 (K2) Kandydat wyjaśnia korzyści wynikające ze śledzenia powiązań.

FL-1.4.5 (K2) Kandydat porównuje poszczególne role występujące w testowaniu.

### 1.5 Niezbędne umiejętności i dobre praktyki w dziedzinie testowania

FL-1.5.1 (K2) Kandydat podaje przykłady ogólnych umiejętności wymaganych w testowaniu.

FL-1.5.2 (K1) Kandydat pamięta, jakie są zalety podejścia „cały zespół”.

FL-1.5.3 (K2) Kandydat omawia korzyści i wady niezależności testowania.

## 1.1. Co to jest testowanie?

Systemy oprogramowania są nieodłączną częścią naszego codziennego życia. Jednocześnie większość z nas miała zapewne do czynienia z oprogramowaniem, które nie zadziałało tak, jak powinno. Nieprawidłowe funkcjonowanie oprogramowania może powodować wiele problemów, w tym straty finansowe, stratę czasu, utratę reputacji firmy, a w skrajnych przypadkach nawet utratę zdrowia lub życia. Odpowiedzią na ten problem jest właśnie testowanie oprogramowania, które pozwala ocenić jego jakość i zmniejszyć ryzyko wystąpienia awarii podczas eksploatacji.

Testowanie oprogramowania to zbiór czynności mających na celu wykrycie defektów i dokonanie oceny jakości artefaktów związanych z oprogramowaniem. W trakcie testowania artefakty te są nazywane przedmiotami testów. Powszechnie panuje błędne przekonanie, że testowanie polega wyłącznie na wykonywaniu testów, czyli uruchamianiu oprogramowania i sprawdzaniu uzyskanych rezultatów. W rzeczywistości jednak testowanie oprogramowania obejmuje również inne czynności i musi być dopasowane do cyklu wytwarzania oprogramowania (patrz rozdział 2).

Inne nieporozumienie polega na postrzeganiu testowania jako czynności skupionej wyłącznie na weryfikowaniu przedmiotu testów. Chociaż w ramach testowania rzeczywiście sprawdza się, czy system spełnia wyspecyfikowane wymagania, to jednak przeprowadza się również walidację, której zadaniem jest sprawdzenie, czy system odpowiada na potrzeby użytkowników i innych interesariuszy w swoim środowisku produkcyjnym.

Testowanie może mieć charakter dynamiczny lub statyczny. Testowanie dynamiczne wiąże się z uruchamianiem oprogramowania, natomiast testowanie statyczne polega na wykonywaniu przeglądów (patrz rozdział 3) i przeprowadzaniu analizy statycznej. W testowaniu dynamicznym do wyprowadzania przypadków testowych używa się różnych technik testowania i podejść do testowania (patrz rozdział 4).

Testowanie nie jest wyłącznie czynnością o charakterze technicznym, ponieważ wymaga również właściwego planowania, zarządzania, szacowania, monitorowania i nadzoru (patrz rozdział 5).

Testerzy korzystają z narzędzi (patrz rozdział 6), ale należy pamiętać, że testowanie to w dużej mierze praca umysłowa, której wykonywanie wymaga posiadania specjalistycznej wiedzy, korzystania z umiejętności analitycznych oraz krytycznego i systemowego myślenia (Myers 2011, Roman 2018).

Więcej informacji na temat pojęć związanych z testowaniem oprogramowania zawiera standard ISO/IEC/IEEE 29119-1.

### 1.1.1. Cele testów

Typowe cele testów to:

- dokonywanie oceny produktów pracy, takich jak wymagania, historyjki użytkownika, projekty, kod;
- powodowanie awarii i znajdowanie defektów;
- zapewnienie wymaganego pokrycia przedmiotu testów;
- obniżanie poziomu ryzyka związanego z niedostateczną jakością oprogramowania;
- sprawdzanie, czy zostały spełnione wyspecyfikowane wymagania;
- sprawdzanie, czy przedmiot testów spełnia wymagania umowne, prawne i regulacyjne;
- dostarczanie interesariuszom informacji niezbędnych do podejmowania świadomych decyzji;

- budowanie zaufania do jakości przedmiotu testów;
- sprawdzanie, czy przedmiot testów jest kompletny i działa zgodnie z oczekiwaniami interesariuszy.

Cele testowania mogą różnić się w zależności od kontekstu, w tym od testowanego produktu pracy, poziomu testów, ryzyk, przyjętego cyklu wytwarzania oprogramowania oraz kwestii związanych z kontekstem biznesowym, takich jak struktura przedsiębiorstwa, uwarunkowania konkurencyjne czy czas wprowadzania produktu na rynek.

### 1.1.2. Testowanie a debugowanie

Testowanie i debugowanie to dwie różne czynności. Testowanie pozwala wywołać awarie, które są skutkiem defektów w oprogramowaniu (testowanie dynamiczne), lub znaleźć defekty bezpośrednio w przedmiocie testów (testowanie statyczne).

Gdy w ramach testowania dynamicznego (patrz rozdział 4) zostanie wywołana awaria, rozpoczyna się debugowanie, którego celem jest znalezienie przyczyn danej awarii (defektów), a następnie ich przeanalizowanie i wyeliminowanie. Typowy proces debugowania stosowany w takich przypadkach obejmuje:

- odtworzenie awarii;
- przeprowadzenie diagnozy (tj. znalezienie podstawowej przyczyny);
- usunięcie przyczyny.

Następnie wykonywane jest testowanie potwierdzające, które pozwala sprawdzić, czy wprowadzone poprawki doprowadziły do rozwiązania problemu. W optymalnych warunkach testowanie potwierdzające wykonuje osoba, która wcześniej przeprowadzała początkowy test. W dalszej kolejności można również wykonać testowanie regresji, aby upewnić się, że wprowadzone poprawki nie powodują awarii w innych obszarach przedmiotu testów (więcej informacji na temat testowania potwierdzającego i testowania regresji zawiera sekcja 2.2.3).

W przypadku wykrycia defektu podczas testowania statycznego, debugowanie obejmuje usunięcie takiego defektu. Nie ma przy tym potrzeby odtwarzania ani diagnozowania problemu, ponieważ testowanie statyczne pozwala wykrywać defekty w sposób bezpośredni i nie może powodować awarii (patrz rozdział 3).

## 1.2. Dlaczego testowanie jest niezbędne?

Testowanie — będące formą kontroli jakości — pomaga osiągnąć uzgodnione cele w wyznaczonym zakresie i czasie, z zachowaniem ustalonego poziomu jakości oraz w granicach przyjętego budżetu. Wkład testowania w powodzenie przedsięwzięcia nie powinien przy tym ograniczać się wyłącznie do działań zespołu testowego. Każdy interesariusz może również wykorzystać swoje umiejętności w dziedzinie testowania, aby przyczynić się do pomyślnej realizacji projektu. Przetestowanie modułów, systemów i związanej z nimi dokumentacji pozwala zidentyfikować defekty w oprogramowaniu.

### 1.2.1. Znaczenie testowania dla powodzenia projektu

Testowanie pozwala w opłacalny sposób wykryć defekty, które można następnie usunąć poprzez debugowanie (będące odrębną czynnością niewchodzącą w zakres samego testowania). Tym samym testowanie pośrednio przyczynia się do podniesienia jakości przedmiotów testów.

Testowanie umożliwia bezpośrednią ocenę jakości przedmiotu testów na różnych etapach cyklu wytwarzania oprogramowania. Czynności związane z testowaniem są wykonywane w ramach szerszych działań związanych z zarządzaniem projektem i pomagają w podejmowaniu decyzji o przejściu do kolejnego etapu cyklu wytwarzania oprogramowania, na przykład o przekazaniu oprogramowania do eksploatacji.

Testowanie stwarza użytkownikom okazję do pośredniego wpływania na przebieg projektu wytwarzania oprogramowania, ponieważ testerzy dbają o to, aby ich znajomość potrzeb użytkowników była uwzględniana na wszystkich etapach cyklu wytwarzania. Alternatywą byłoby zaproszenie reprezentatywnej grupy użytkowników do udziału w projekcie wytwarzania oprogramowania, co zwykle nie jest możliwe z uwagi na duże koszty oraz brak dostępności odpowiednich osób.

Testowanie może być również niezbędne do spełnienia wymagań wynikających z umów, przepisów prawa lub norm/standardów.

### 1.2.2. Testowanie a zapewnienie jakości

Testowanie jest często utożsamiane z zapewnieniem jakości, ale w rzeczywistości są to dwa oddzielne procesy. Testowanie jest formą kontroli jakości.

Kontrola jakości to podejście ukierunkowane na produkt i mające charakter korekcyjny, które skupia się na wykonywaniu czynności umożliwiających osiągnięcie odpowiedniego poziomu jakości. Testowanie jest ważną formą kontroli jakości — obok metod formalnych (takich jak weryfikacja modelowa czy kontrola poprawności), symulacji i prototypowania.

Z kolei zapewnienie jakości to podejście ukierunkowane na procesy i mające charakter prewencyjny, które skupia się na wdrażaniu i udoskonalaniu procesów. Bazuje ono na założeniu, że poprawne wykonanie dobrze zaprojektowanego procesu przekłada się na wytworzenie dobrego produktu. Zasady zapewnienia jakości mają zastosowanie zarówno do wytwarzania, jak i do testowania oprogramowania, a odpowiedzialność za ich przestrzeganie ponoszą wszystkie osoby zaangażowane w projekt.

Wyniki testów są wykorzystywane w obu przypadkach: w kontekście kontroli jakości pomagają w usuwaniu defektów, a w kontekście zapewnienia jakości dostarczają informacji zwrotnych na temat tego, na ile prawidłowo przebiegają procesy wytwarzania i testowania oprogramowania.

### 1.2.3. Pomyłki, defekty, awarie i podstawowe przyczyny

Na skutek pomyłki (błędu) człowieka mogą powstać defekty (inaczej zwane usterkami lub pluskwami), które w dalszej kolejności mogą prowadzić do wystąpienia awarii. Ludzie popełniają pomyłki z różnych przyczyn, takich jak presja czasu, złożoność produktów pracy, procesów, infrastruktury lub interakcji bądź po prostu zmęczenie lub brak należytego przeszkolenia.

Defekty mogą występować w dokumentacji (np. w specyfikacji wymagań lub w skrypcie testowym), w kodzie źródłowym lub w artefaktach pomocniczych (takich jak plik kompilacji). Niewykrycie defektów

w artefaktach powstałych na wcześniejszych etapach cyklu wytwarzania oprogramowania często prowadzi do powstania wadliwych artefaktów na dalszych etapach tego cyklu. Wykonanie kodu zawierającego defekt może doprowadzić do sytuacji, w której system nie wykona zamierzonej operacji lub wykona operację niezamierzoną, czyli do awarii. Niektóre defekty zawsze powodują awarię w przypadku wykonania wadliwego kodu, a inne — tylko w określonych okolicznościach. Może się również zdarzyć, że dany defekt nigdy nie spowoduje awarii.

Pomyłki i defekty nie są jedynymi przyczynami awarii. Awarie mogą być również spowodowane warunkami środowiskowymi, takimi jak promieniowanie lub pole elektromagnetyczne powodujące nieprawidłowe działanie oprogramowania wbudowanego (ang. *firmware*).

Zasadniczy czynnik powodujący wystąpienie problemu (np. sytuacja prowadząca do pomyłki) jest nazywany podstawową przyczyną. Podstawowe przyczyny ustala się w toku tzw. analizy przyczyny podstawowej, którą przeprowadza się zwykle po wystąpieniu awarii lub zidentyfikowaniu defektu. Uważa się, że podjęcie odpowiednich działań w stosunku do podstawowej przyczyny (na przykład jej usunięcie) pozwala zapobiec powstaniu kolejnych podobnych awarii lub defektów, a przynajmniej zmniejszyć ich częstotliwość.

## 1.3. Zasady testowania

Na przestrzeni lat zaproponowano cały szereg zasad testowania, które dostarczają ogólnych wskazówek mających zastosowanie do wszystkich rodzajów testowania. W niniejszym sylabusie opisano siedem takich zasad.

**1. Testowanie ujawnia defekty, ale nie może dowieść ich braku.** Testowanie może wykazać obecność defektów w przedmiocie testów, ale nie może dowieść, że jest on od nich wolny (Buxton 1970). Tym samym testowanie zmniejsza prawdopodobieństwo, że w przedmiocie testów pozostaną niewykryte defekty, ale sam fakt niewykrycia defektów nie stanowi dowodu poprawności testowanego produktu.

**2. Testowanie gruntowne jest niemożliwe.** Przetestowanie wszystkiego jest możliwe tylko w najprostszych przypadkach (Manna 1978). W związku z tym wysiłki związane z testowaniem powinny być ukierunkowane raczej na stosowanie technik testowania (patrz rozdział 4), ustalanie priorytetów przypadków testowych (patrz sekcja 5.1.5) oraz testowanie oparte na ryzyku (patrz podrozdział 5.2).

**3. Wczesne testowanie oszczędza czas i pieniądze.** Defekty usunięte na wczesnym etapie procesu nie powodują powstania kolejnych defektów w pochodnych produktach pracy. Przekłada się to na obniżenie kosztu jakości, ponieważ dzięki temu zmniejsza się liczba awarii występujących na późniejszych etapach cyklu wytwarzania oprogramowania (Boehm 1981). Aby wcześnie wykryć defekty, należy jak najszybciej rozpocząć zarówno testowanie statyczne (patrz rozdział 3), jak i testowanie dynamiczne (patrz rozdział 4).

**4. Defekty mogą się kumulować.** Zwykle większość wykrytych defektów lub większość awarii występujących w fazie eksploatacji powstaje lub ma swoje źródło w niewielkiej liczbie modułów systemu (Enders 1975), co jest ilustracją tzw. zasady Pareto. Dlatego przewidywane skupiska defektów i skupiska defektów faktycznie zaobserwowane na etapie testowania lub eksploatacji są ważnym elementem testowania opartego na ryzyku (patrz podrozdział 5.2).

**5. Testy ulegają zużyciu.** Wielokrotne powtarzanie tych samych testów prowadzi do spadku ich skuteczności w wykrywaniu nowych defektów (Beizer 1990). Przezwyciężenie tego problemu może wymagać zmodyfikowania dotychczasowych testów i danych testowych oraz napisania nowych testów. W niektórych przypadkach powtarzanie tych samych testów może być jednak korzystne, czego przykładem jest automatyczne testowanie regresji (patrz sekcja 2.2.3).

**6. Testowanie zależy od kontekstu.** Nie ma jednego uniwersalnego podejścia do testowania. Testowanie wykonuje się w różny sposób w różnych kontekstach (Kaner 2011).

**7. Przekonanie o braku defektów jest błędem.** Przekonanie, że sama weryfikacja oprogramowania zapewni pomyślne wdrożenie systemu, jest błędne. Nawet bardzo dokładne przetestowanie wszystkich wyspecyfikowanych wymagań i usunięcie wszystkich znalezionych defektów nie chronią przed zbudowaniem systemu, który nie zaspokoi potrzeb użytkowników i nie spełni ich oczekiwań, nie pomoże klientowi w osiągnięciu celów biznesowych oraz będzie miał gorsze parametry od konkurencyjnych rozwiązań. Dlatego oprócz weryfikacji należy również przeprowadzać walidację (Boehm 1981).

## 1.4. Czynności testowe, testalia i role związane z testami

Testowanie zależy od kontekstu, ale na poziomie ogólnym można wyróżnić pewne typowe grupy czynności testowych, których pominięcie zmniejsza prawdopodobieństwo osiągnięcia celów testów. Powyższe grupy czynności testowych składają się na proces testowy. Dobór procesu testowego do konkretnej sytuacji zależy od wielu czynników. Decyzję co do tego, które czynności testowe mają być uwzględnione w tym procesie oraz jak i kiedy mają być realizowane, podejmuje się zwykle na etapie planowania testów i z uwzględnieniem konkretnej sytuacji (patrz podrozdział 5.1).

W kolejnych sekcjach opisano ogólne aspekty procesu testowego: czynności i zadania testowe, znaczenie kontekstu, testalia, śledzenie powiązań między podstawą testów a testaliami oraz role występujące w testowaniu.

Więcej informacji na temat procesów testowych zawiera standard ISO/IEC/IEEE 29119-2.

### 1.4.1. Czynności i zadania testowe

W procesie testowym wyróżnia się główne grupy czynności opisane poniżej. Chociaż wiele z tych czynności może sprawiać wrażenie logicznie uszeregowanych, często są one realizowane metodą iteracyjną lub w sposób równoległy. Ponadto zwykle wymagają dostosowania do potrzeb konkretnego systemu i projektu.

**Planowanie testów** polega na zdefiniowaniu celów testów, a następnie dokonaniu wyboru podejścia, które pozwoli najskuteczniej osiągnąć te cele w ramach ograniczeń narzuconych przez ogólny kontekst. Kwestię planowania testów omówiono szczegółowo w podrozdziale 5.1.

**Monitorowanie testów i nadzór nad testami.** Monitorowanie testów polega na ciągłym sprawdzaniu wszystkich czynności testowych i porównywaniu rzeczywistego postępu z założeniami przyjętymi w planie, a nadzór nad testami polega na podejmowaniu działań, które są niezbędne do osiągnięcia celów testowania. Kwestię monitorowania testów i nadzoru nad testami objaśniono dokładniej w podrozdziale 5.3.

**Analiza testów** polega na przeanalizowaniu podstawy testów w celu zidentyfikowania testowalnych cech oraz zdefiniowania i określenia priorytetów związanych z nimi warunków testowych, z uwzględnieniem występujących w danym przypadku ryzyk i poziomów ryzyka (patrz podrozdział 5.2). Ponadto czynność ta obejmuje przeanalizowanie podstawy testów i przedmiotów testów w celu zidentyfikowania ewentualnych defektów, jakie mogą w nich występować, oraz dokonanie oceny ich testowalności. Do przeprowadzania analizy testów często wykorzystuje się techniki testowania (patrz rozdział 4). Analiza testów służy do ustalenia tego, „co należy przetestować” (w kategoriach mierzalnych kryteriów pokrycia).

**Projektowanie testów** polega na przekształceniu warunków testowych w przypadki testowe i inne testalia (np. karty opisu testów). Czynność ta zwykle wiąże się ze zidentyfikowaniem elementów pokrycia, które dostarczają wskazówek przy określaniu danych wejściowych dla przypadków testowych. Przy wykonywaniu tej czynności można wykorzystać techniki testowania (patrz rozdział 4). Ponadto projektowanie testów obejmuje określenie wymagań dotyczących danych testowych, zaprojektowanie środowiska testowego oraz zidentyfikowanie innych niezbędnych narzędzi i elementów infrastruktury. Reasumując, projektowanie testów odpowiada na pytanie o to, „jak należy testować".

**Implementacja testów** polega na utworzeniu lub pozyskaniu testaliów niezbędnych do wykonywania testów (np. danych testowych). Czynność ta może obejmować uszeregowanie przypadków testowych w ramach procedur testowych, a często również połączenie ich w zestawy testowe. Ponadto grupa ta obejmuje tworzenie skryptów testów manualnych i automatycznych, a także szeregowanie procedur testowych według priorytetów i porządkowanie ich w ramach harmonogramu wykonywania testów w sposób zapewniający efektywne wykonanie testów (patrz sekcja 5.1.5). Ostatnim elementem jest zbudowanie środowiska testowego i sprawdzenie, czy zostało ono poprawnie skonfigurowane.

**Wykonywanie testów** polega na uruchamianiu testów zgodnie z harmonogramem wykonywania testów (czyli na wykonywaniu przebiegów testów), przy czym może się to odbywać manualnie lub automatycznie. Wykonywanie testów może przybierać wiele form, takich jak ciągłe testowanie lub sesje testowania w parach. Rzeczywiste rezultaty testów są porównywane z rezultatami oczekiwanymi. Ponadto wyniki testów są rejestrowane, a ewentualne anomalie są analizowane w celu ustalenia ich prawdopodobnych przyczyn. Analiza ta umożliwia zgłaszanie anomalii na podstawie zaobserwowanych awarii (patrz podrozdział 5.5).

**Ukończenie testów** obejmuje czynności, które są zwykle wykonywane w momencie osiągnięcia kamieni milowych projektu (takich jak przekazanie do eksploatacji, zakończenie iteracji lub ukończenie testów danego poziomu) w odniesieniu do wszelkich nieusuniętych defektów, zgłoszonych żądań zmian bądź utworzonego backlogu produktu. Wszelkie testalia, które mogą okazać się przydatne w przyszłości, są identyfikowane i archiwizowane bądź przekazywane odpowiednim zespołom. Środowisko testowe jest zamykane w uzgodnionym stanie, a czynności testowe są analizowane w celu sformułowania wniosków i zidentyfikowania udoskonaleń, które będzie można wprowadzić w przypadku przyszłych iteracji, wydań lub projektów (patrz sekcja 2.1.6). Ostatnim elementem jest stworzenie sumarycznego raportu z testów i przekazanie go interesariuszom.

### 1.4.2. Proces testowy w kontekście

Testowanie nie odbywa się w izolacji. Czynności testowe są integralnym elementem procesów wytwarzania oprogramowania realizowanych w danej organizacji. Co więcej, testowanie jest finansowane także przez interesariuszy, a jego nadrzędnym celem jest zaspokajanie ich potrzeb biznesowych. Z tego względu sposób, w jaki wykonywane jest testowanie, zależy od kontekstu, na który składają się między innymi następujące czynniki:

- interesariusze (w tym ich potrzeby, oczekiwania, wymagania, gotowość do współpracy itd.);
- członkowie zespołu (w tym ich umiejętności, wiedza, doświadczenie, dostępność, potrzeby w zakresie szkoleń itd.);
- dziedzina biznesowa (krytyczność przedmiotu testów, zidentyfikowane ryzyka, potrzeby rynku, konkretne uregulowania prawne itd.);
- czynniki techniczne (rodzaj oprogramowania, architektura produktu, zastosowana technologia itd.);

- ograniczenia związane z projektem (zakres, terminy, budżet, zasoby itd.);
- czynniki organizacyjne (struktura organizacyjna, istniejące polityki, przyjęte praktyki itd.);
- cykl wytwarzania oprogramowania (praktyki w dziedzinie inżynierii oprogramowania, metody wytwarzania itd.);
- narzędzia (dostępność, łatwość używania, zgodność itd.).

Powyższe czynniki mają wpływ na wiele kwestii związanych z testami, w tym na strategię testów, stosowane techniki testowania, stopień automatyzacji testów, wymagany poziom pokrycia, szczegółowość dokumentacji testów, raportowanie itd.

### 1.4.3. Testalia

W wyniku wykonywania czynności testowych opisanych w sekcji 1.4.1 powstają produkty pracy nazywane testaliami. Sposób wytwarzania, kształtowania i porządkowania takich produktów pracy oraz zarządzania nimi, a także nadawane im nazwy różnią się znacznie w poszczególnych organizacjach. Dlatego do zapewnienia spójności i integralności produktów pracy niezbędne jest właściwe zarządzanie konfiguracją (patrz podrozdział 5.4). Poniżej opisano wybrane kategorie produktów pracy.

- **Produkty pracy związane z planowaniem testów** obejmują: plan testów, harmonogram testów, rejestr ryzyk oraz kryteria wejścia i wyjścia (patrz podrozdział 5.1). Rejestr ryzyk zawiera wykaz ryzyk wraz z informacjami o prawdopodobieństwie, wpływie i sposobach łagodzenia każdego z nich (patrz podrozdział 5.2). Harmonogram testów, rejestr ryzyk oraz kryteria wejścia i wyjścia są często elementami planu testów.
- **Produkty pracy związane z monitorowaniem testów i nadzorem nad testami** obejmują: raporty o postępie testów (patrz sekcja 5.3.2), dokumentację dotyczącą dyrektyw nadzoru (patrz podrozdział 5.3) oraz informacje o ryzyku (patrz podrozdział 5.2).
- **Produkty pracy związane z analizą testów** to między innymi: uszeregowane według priorytetów warunki testowe (np. kryteria akceptacji, patrz sekcja 4.5.2) i raporty o defektach dotyczące defektów w podstawie testów (jeśli nie zostały one natychmiast usunięte).
- **Produkty pracy związane z projektowaniem testów** obejmują: uszeregowane według priorytetów przypadki testowe, karty opisu testów, elementy pokrycia oraz wymagania dotyczące danych testowych i środowiska testowego.
- **Produkty pracy związane z implementacją testów** obejmują: procedury testowe, skrypty testów automatycznych, zestawy testowe, dane testowe, harmonogram wykonywania testów oraz elementy środowiska testowego. Przykładowe elementy środowiska testowego to między innymi: zaślepki, sterowniki, symulatory i wirtualizacja usług.
- **Produkty pracy związane z wykonywaniem testów** obejmują: dzienniki testów i raporty o defektach (patrz podrozdział 5.5).
- **Produkty pracy związane z ukończeniem testów** to między innymi: sumaryczny raport z testów (patrz sekcja 5.3.2), lista czynności do wykonania mających na celu wprowadzenie udoskonaleń w kolejnych projektach lub iteracjach, udokumentowane wnioski oraz żądania zmian (np. w formie backlogu produktu).

### 1.4.4. Śledzenie powiązań między podstawą testów a testaliami

W celu zapewnienia skutecznego monitorowania testów i nadzoru nad testami należy stworzyć i utrzymywać mechanizm śledzenia powiązań między elementami podstawy testów a związanymi z nimi testaliami (np. warunkami testowymi, ryzykami lub przypadkami testowymi), wynikami testów i wykrytymi defektami przez cały czas trwania procesu testowego.

Dokładne śledzenie powiązań ułatwia ocenę pokrycia testowego, dlatego bardzo przydatne jest zdefiniowanie w podstawie testów mierzalnych kryteriów pokrycia. Kryteria takie mogą pełnić funkcję kluczowych wskaźników wydajności (ang. *key performance indicator — KPI*) sprzyjających wykonywaniu określonych czynności i pozwalających określić stopień realizacji celów testów (patrz sekcja 1.1.1). Na przykład:

- Śledzenie powiązań między przypadkami testowymi a wymaganiami pozwala potwierdzić, że wymagania zostały pokryte przez przypadki testowe.
- Śledzenie powiązań między wynikami testów a ryzykami umożliwia ocenę poziomu ryzyka rezydualnego (resztkowego) w przedmiocie testów.

Sprawne śledzenie powiązań umożliwia nie tylko ocenę pokrycia testowego — odpowiedni mechanizm śledzenia pozwala również ustalać wpływ zmian oraz ułatwia przeprowadzanie audytów testów i spełnianie kryteriów związanych z zarządzaniem w obszarze IT. Ponadto uwzględnienie statusu elementów podstawy testów umożliwia tworzenie bardziej zrozumiałych raportów o postępie testów i sumarycznych raportów z testów, co z kolei ułatwia przekazywanie interesariuszom informacji o aspektach technicznych testowania w zrozumiałej dla nich formie. Kolejną zaletą śledzenia jest możliwość udzielania informacji potrzebnych do oceny jakości produktów, wydajności procesów i postępu w realizacji projektu z punktu widzenia celów biznesowych.

### 1.4.5. Role w procesie testowania

W niniejszym sylabusie omówiono dwie zasadnicze role występujące w testowaniu: rolę związaną z zarządzaniem testami i rolę związaną z testowaniem. Czynności i zadania wyznaczone osobom pełniącym obie role zależą od takich czynników jak kontekst projektu i produktu, umiejętności konkretnych osób oraz specyfika organizacji.

Osoba pełniąca rolę związaną z zarządzaniem testami ponosi ogólną odpowiedzialność za proces testowy i pracę zespołu testowego oraz za kierowanie przebiegiem czynności testowych. Rola ta polega głównie na wykonywaniu czynności związanych z planowaniem testów, monitorowaniem testów, nadzorem nad testami oraz ukończeniem testów. Sposób wykonywania obowiązków wynikających z roli związanej z zarządzaniem testami zależy od kontekstu. Na przykład w ramach zwinnego wytwarzania oprogramowania niektóre z zadań związanych z zarządzaniem testami może wykonywać zespół zwinny, a zadania obejmujące swoim zasięgiem kilka zespołów lub całą organizację mogą wykonywać kierownicy testów spoza zespołu tworzącego oprogramowanie.

Osoba pełniąca rolę związaną z testowaniem ponosi ogólną odpowiedzialność za aspekty techniczne testowania. Rola ta polega głównie na podejmowaniu działań związanych z analizą, projektowaniem, implementacją i wykonywaniem testów.

W zależności od etapu prac powyższe role mogą pełnić różne osoby — na przykład rolę związaną z zarządzaniem testami może pełnić lider zespołu, kierownik testów, kierownik zespołu tworzącego

oprogramowanie itd. Ponadto ta sama osoba może pełnić jednocześnie rolę związaną z testowaniem i rolę związaną z zarządzaniem testami.

## 1.5. Niezbędne umiejętności i dobre praktyki w dziedzinie testowania

Umiejętność to zdolność do prawidłowego wykonywania określonej czynności wynikająca z wiedzy, praktyki i osobistych uzdolnień. Aby móc właściwie wykonywać swoją pracę, dobry tester musi posiadać pewne niezbędne umiejętności, takie jak umiejętność sprawnej pracy w zespole czy wykonywania testów na różnych poziomach niezależności.

### 1.5.1. Ogólne umiejętności wymagane w związku z testowaniem

W przypadku testerów szczególnie istotne są następujące umiejętności o charakterze ogólnym:

- wiedza w dziedzinie testowania (umożliwiająca zwiększanie skuteczności testowania np. poprzez zastosowanie technik testowania);
- staranność, ostrożność, ciekawość, dbałość o szczegóły i metodyczność (niezbędne do identyfikowania defektów, zwłaszcza defektów trudnych do wykrycia);
- umiejętności komunikacyjne oraz umiejętność aktywnego słuchania i pracy w zespole (pozwalające sprawnie komunikować się ze wszystkimi interesariuszami, w zrozumiały sposób przekazywać informacje innym osobom oraz zgłaszać i omawiać defekty);
- umiejętność analitycznego i krytycznego myślenia oraz kreatywność (umożliwiające zwiększanie skuteczności testowania);
- wiedza techniczna (pozwalająca zwiększyć efektywność testowania np. poprzez korzystanie z odpowiednich narzędzi testowych);
- wiedza merytoryczna (pozwalająca zrozumieć użytkowników i przedstawicieli jednostek biznesowych oraz sprawnie się z nimi porozumiewać).

Z uwagi na specyfikę pracy testera istotnym problemem może być powszechna u ludzi skłonność do obwiniania osoby przynoszącej złe wiadomości, dlatego tak ważne są w tym przypadku umiejętności komunikacyjne. Informowanie o wynikach testów może być odbierane jako krytyka produktu i jego autora, a zjawisko psychologiczne zwane efektem potwierdzenia (ang. *confirmation bias*) może utrudniać zaakceptowanie informacji sprzecznych z dotychczasowymi przekonaniami. Ponadto niektóre osoby mogą postrzegać testowanie jako czynność destrukcyjną, nawet jeśli przyczynia się ono wydatnie do powodzenia projektu i podnoszenia jakości produktów. Aby przełamać ten stereotyp, należy przekazywać informacje o defektach i awariach w sposób jak najbardziej konstruktywny.

### 1.5.2. Podejście „cały zespół”

Jedną z ważnych umiejętności, jaką powinien dysponować tester, jest umiejętność sprawnej pracy zespołowej i działania na rzecz realizacji celów zespołu. Podejście „cały zespół” (ang. *whole-team approach*), które ma swoje początki w modelu programowania ekstremalnego (ang. *eXtreme Programming — XP*) (patrz podrozdział 2.1), opiera się właśnie na tej umiejętności.

W ramach podejścia „cały zespół” każdy członek zespołu, który dysponuje niezbędną wiedzą i umiejętnościami, może wykonywać dowolne zadania, a odpowiedzialność za jakość spoczywa w równym

stopniu na wszystkich. Członkowie zespołu pracują w tym samym obszarze roboczym (fizycznym lub wirtualnym), ponieważ sprzyja to wymianie informacji i współdziałaniu. Podejście takie zwiększa dynamikę pracy zespołowej, usprawnia wymianę informacji i współpracę w zespole oraz tworzy efekt synergii, ponieważ umożliwia wykorzystanie różnych kombinacji umiejętności na rzecz realizacji projektu.

Testerzy ściśle współpracują z innymi członkami zespołu, aby zagwarantować osiągnięcie wymaganych poziomów jakości. Obejmuje to również współpracę z przedstawicielami jednostek biznesowych w celu stworzenia odpowiednich testów akceptacyjnych oraz z programistami w celu uzgodnienia strategii testów i podjęcia decyzji co do sposobu automatyzacji testów. W ten sposób testerzy mogą przekazywać wiedzę w dziedzinie testowania innym członkom zespołu oraz wpływać na proces wytwarzania produktu.

Należy jednak pamiętać, że w pewnych sytuacjach zaangażowanie całego zespołu może nie być optymalnym rozwiązaniem. Dotyczy to na przykład systemów krytycznych ze względów bezpieczeństwa, w przypadku których może być wymagany wysoki poziom niezależności testów.

### 1.5.3. Niezależność testowania

Pewien stopień niezależności często zwiększa skuteczność wykrywania defektów, ponieważ działania autora i testera mogą być obarczone różnymi błędami poznawczymi (por. Salman 1995). Jednocześnie jednak niezależność nie zastępuje znajomości produktu, a programiści mogą efektywnie wykrywać wiele defektów w tworzonym przez siebie kodzie.

Produkty pracy mogą być testowane przez autora (brak niezależności), przez innych członków zespołu autora (pewien stopień niezależności), przez testerów spoza zespołu autora w obrębie danej organizacji (wysoki poziom niezależności) lub przez testerów spoza organizacji (bardzo wysoki poziom niezależności). W większości projektów najlepiej sprawdza się przeprowadzanie testów na wielu poziomach niezależności (przykładem może być sytuacja, w której programiści wykonują testowanie modułowe i testowanie integracji modułów, zespół testowy wykonuje testowanie systemowe i testowanie integracji systemów, a przedstawiciele jednostek biznesowych wykonują testowanie akceptacyjne).

Główną korzyścią wynikającą z niezależności testowania jest prawdopodobieństwo wykrycia przez niezależnych testerów innego rodzaju awarii i defektów niż te wykryte przez programistów ze względu na różne doświadczenia, techniczne punkty widzenia i błędy poznawcze. Ponadto niezależny tester może zweryfikować, zakwestionować lub obalić założenia przyjęte przez interesariuszy na etapie specyfikowania i implementacji systemu.

Nie wolno też jednak zapominać o pewnych wadach. Odizolowanie niezależnych testerów od zespołu deweloperskiego może prowadzić do braku współpracy, problemów z wymianą informacji, a nawet konfliktu z tym zespołem. Sytuacja taka rodzi niebezpieczeństwo utraty przez programistów poczucia odpowiedzialności za jakość oraz stwarza ryzyko, że niezależni testerzy zostaną potraktowani jako wąskie gardło i obarczeni winą za nieterminowe przekazanie produktu do eksploatacji.

# 2. Testowanie w cyklu wytwarzania oprogramowania — 130 minut

**Słowa kluczowe**

poziom testów, przedmiot testów, przesunięcie w lewo, testowanie akceptacyjne, testowanie białoskrzynkowe, testowanie czarnoskrzynkowe, testowanie funkcjonalne, testowanie integracji modułów, testowanie integracji systemów, testowanie integracyjne, testowanie modułowe, testowanie niefunkcjonalne, testowanie pielęgnacyjne, testowanie potwierdzające, testowanie regresji, testowanie systemowe, typ testów

**Cele nauczania w rozdziale 2:**

### 2.1 Testowanie w kontekście cyklu wytwarzania oprogramowania

FL-2.1.1 (K2) Kandydat wyjaśnia wpływ wybranego modelu cyklu wytwarzania oprogramowania na testowanie.

FL-2.1.2 (K1) Kandydat pamięta dobre praktyki testowania mające zastosowanie do wszystkich modeli cyklu wytwarzania oprogramowania.

FL-2.1.3 (K1) Kandydat podaje przykłady podejść typu „najpierw test” w kontekście wytwarzania oprogramowania.

FL-2.1.4 (K2) Kandydat podsumowuje, w jaki sposób metodyka DevOps może wpłynąć na testowanie.

FL-2.1.5 (K2) Kandydat wyjaśnia, na czym polega przesunięcie w lewo.

FL-2.1.6 (K2) Kandydat wyjaśnia, w jaki sposób retrospektywy mogą posłużyć jako mechanizmy doskonalenia procesów.

### 2.2 Poziomy testów i typy testów

FL-2.2.1 (K2) Kandydat rozróżnia poszczególne poziomy testów.

FL-2.2.2 (K2) Kandydat rozróżnia poszczególne typy testów.

FL-2.2.3 (K2) Kandydat odróżnia testowanie potwierdzające od testowania regresji.

### 2.3 Testowanie pielęgnacyjne

FL-2.3.1 (K2) Kandydat podsumowuje testowanie pielęgnacyjne i zdarzenia je wyzwalające.

# 2.1. Testowanie w kontekście modelu cyklu wytwarzania oprogramowania

Model cyklu wytwarzania oprogramowania (ang. *software development lifecycle — SDLC*) stanowi abstrakcyjne, ogólne odwzorowanie procesu wytwarzania oprogramowania. Model ten określa wzajemne relacje — zarówno logiczne, jak i chronologiczne — między poszczególnymi fazami wytwarzania oprogramowania oraz rodzajami czynności wykonywanych w ramach tego procesu. Przykładami modeli cyklu wytwarzania oprogramowania są między innymi: sekwencyjne modele wytwarzania oprogramowania (np. model kaskadowy - ang. *waterfall* - lub model V), iteracyjne modele wytwarzania oprogramowania (np. model spiralny lub prototypowanie) oraz przyrostowe modele wytwarzania oprogramowania (np. model *Unified Process*).

Niektóre czynności wykonywane w ramach procesów wytwarzania oprogramowania można również opisać przy użyciu bardziej szczegółowych metod wytwarzania oprogramowania i praktyk zwinnych. Przykładami są między innymi: wytwarzanie sterowane testami akceptacyjnymi (ang. *acceptance test-driven development — ATDD*), wytwarzanie sterowane zachowaniem (ang. *behavior-driven development — BDD*), projektowanie oparte na domenie (ang. *domain-driven design — DDD*), programowanie ekstremalne (ang. *eXtreme Programming — XP*), wytwarzanie oparte na cechach (ang. *feature-driven development — FDD*), Kanban, Lean IT, Scrum oraz wytwarzanie sterowane testami (ang. *test-driven development — TDD*).

## 2.1.1. Wpływ cyklu wytwarzania oprogramowania na testowanie

Warunkiem powodzenia procesu testowania jest dopasowanie go do przyjętego cyklu wytwarzania oprogramowania. Wybór modelu cyklu wytwarzania oprogramowania wpływa na:

- zakres i czas wykonywania czynności testowych (np. poziomy testów i typy testów);
- szczegółowość dokumentacji testów;
- wybór technik testowania i podejścia do testowania;
- zakres automatyzacji testów;
- role i obowiązki testera.

W przypadku sekwencyjnych modeli wytwarzania oprogramowania w początkowych fazach procesu testerzy uczestniczą zazwyczaj w przeglądach wymagań, analizie testów oraz projektowaniu testów. Kod wykonywalny powstaje z reguły w późniejszych fazach, co w większości przypadków uniemożliwia przeprowadzenie testowania dynamicznego na wczesnym etapie cyklu wytwarzania oprogramowania.

W niektórych iteracyjnych i przyrostowych modelach wytwarzania zakłada się, że w wyniku każdej iteracji powstaje działający prototyp lub kolejna wersja przyrostowa produktu. Oznacza to, że w każdej iteracji można wykonywać zarówno testowanie statyczne, jak i testowanie dynamiczne na wszystkich poziomach testów. Jednocześnie częste dostarczanie wersji przyrostowych wymaga szybkiego przekazywania informacji zwrotnych i szeroko zakrojonego testowania regresji.

W przypadku zwinnego wytwarzania oprogramowania dopuszcza się wprowadzanie zmian przez cały czas trwania projektu. Z tego powodu w projektach zwinnych wskazane jest tworzenie uproszczonej dokumentacji produktów pracy i stosowanie na dużą skalę automatyzacji testów, co ułatwia testowanie regresji. Ponadto większość testowania manualnego wykonuje się zwykle przy użyciu technik testowania opartych na doświadczeniu (patrz podrozdział 4.4), które nie wymagają wcześniejszego podjęcia szeroko zakrojonych działań związanych z analizą i projektowaniem testów.

### 2.1.2. Model cyklu wytwarzania oprogramowania a dobre praktyki testowania

Poniżej wymieniono kilka dobrych praktyk testowania, które mają zastosowanie do każdego modelu cyklu wytwarzania oprogramowania:

- Do każdej czynności związanej z wytwarzaniem oprogramowania powinna być przypisana odpowiadająca jej czynność testowa, tak aby wszystkie czynności związane z wytwarzaniem oprogramowania podlegały kontroli jakości.
- Poszczególnym poziomom testów (patrz sekcja 2.2.1) powinny odpowiadać konkretne i różne cele testów, co pozwoli zapewnić odpowiednio szeroki zakres testowania, a przy tym uniknąć nadmiarowości.
- Aby zapewnić zgodność z zasadą wczesnego testowania (patrz podrozdział 1.3), analizę i projektowanie testów na potrzeby danego poziomu testów należy rozpocząć w odpowiadającej temu poziomowi fazie cyklu wytwarzania oprogramowania.
- Testerzy powinni uczestniczyć w przeglądach produktów pracy natychmiast po udostępnieniu wersji roboczych odpowiednich dokumentów, tak aby wcześniejsze testowanie i wykrywanie defektów pomagało w realizacji podejścia przesunięcie w lewo (ang. *shift left*; patrz sekcja 2.1.5).

### 2.1.3. Testowanie jako czynnik określający sposób wytwarzania oprogramowania

Wytwarzanie sterowane testami (TDD), wytwarzanie sterowane testami akceptacyjnymi (ATDD) i wytwarzanie sterowane zachowaniem (BDD) to podobne podejścia do wytwarzania oprogramowania, w ramach których testy traktuje się jako czynnik określający sposób prowadzenia prac programistycznych. Każde z tych podejść realizuje zasadę wczesnego testowania (patrz podrozdział 1.3) i podejście przesunięcie w lewo (patrz sekcja 2.1.5), ponieważ testy są definiowane przed rozpoczęciem pisania kodu. Ponadto podejścia te umożliwiają stosowanie iteracyjnego modelu wytwarzania. Poniżej przedstawiono najważniejsze cechy każdego z nich:

Wytwarzanie sterowane testami (TDD):

- Zamiast rozbudowanych mechanizmów projektowania oprogramowania do określania sposobu tworzenia kodu wykorzystywane są przypadki testowe (Beck 2003).
- Najpierw pisane są testy, a dopiero potem powstaje kod umożliwiający ich pomyślne przejście. Następnie testy i kod podlegają refaktoryzacji.

Wytwarzanie sterowane testami akceptacyjnymi (ATDD; patrz sekcja 4.5.3):

- Testy są tworzone na podstawie kryteriów akceptacji w ramach procesu projektowania systemu (Gärtner 2011).
- Testy są pisane przed wytworzeniem części aplikacji, która ma je pomyślnie przejść.

Wytwarzanie sterowane zachowaniem (BDD):

- Pożądane zachowanie aplikacji wyraża się w postaci przypadków testowych napisanych w prostej formie języka naturalnego, która jest zrozumiała dla interesariuszy — zwykle w formacie Given/When/Then (Mając/Kiedy/Wtedy) (Chelimsky 2010).
- Następnie przypadki testowe są automatycznie przekładane na możliwe do wykonania testy.

W ramach wszystkich powyższych podejść testy mogą zostać zarejestrowane w postaci testów automatycznych, co pozwala zapewnić należytą jakość kodu w przypadku przyszłej adaptacji lub refaktoryzacji.

### 2.1.4. Metodyka DevOps a testowanie

DevOps to metodyka organizacyjna, której celem jest uzyskanie efektu synergii poprzez ukierunkowanie działań związanych z wytwarzaniem oprogramowania (w tym z testowaniem) oraz działań związanych z jego eksploatacją na realizację szeregu wspólnych celów. Do wdrożenia tej metodyki niezbędna jest zmiana kultury organizacyjnej polegająca na wypełnieniu luki między strukturami odpowiedzialnymi za wytwarzanie (w tym testowanie) a strukturami odpowiedzialnymi za eksploatację przy jednoczesnym uznaniu, że zadania realizowane przez te struktury są równie istotne. DevOps sprzyja większej autonomii zespołów, szybszemu przekazywaniu informacji zwrotnych, ściślejszej integracji łańcuchów narzędzi oraz stosowaniu praktyk technicznych takich jak ciągła integracja (ang. *continuous integration — CI*) i ciągłe dostarczanie (ang. *continuous delivery — CD*). Dzięki temu zespoły mogą szybciej tworzyć, testować i przekazywać do eksploatacji wysokiej jakości kod z wykorzystaniem potoku dostarczania DevOps (Kim 2016).

Z perspektywy testowania do korzyści ze stosowania metodyki DevOps można zaliczyć:

- szybkie otrzymywanie informacji zwrotnych na temat jakości kodu oraz ewentualnego niekorzystnego wpływu zmian na dotychczasowy kod;
- ciągłą integrację, która sprzyja przesunięciu w lewo w obszarze testowania (patrz sekcja 2.1.5) poprzez zachęcenie programistów do dostarczania wysokiej jakości kodu sprawdzonego testami modułowymi i analizą statyczną;
- promowanie zautomatyzowanych procesów takich jak ciągła integracja i ciągłe dostarczanie, które ułatwiają tworzenie stabilnych środowisk testowych;
- zwiększenie widoczności niefunkcjonalnych charakterystyk jakościowych (np. wydajności lub niezawodności);
- zmniejszenie zapotrzebowania na powtarzalne testowanie manualne dzięki automatyzacji, jaką zapewnia potok dostarczania;
- zmniejszenie ryzyka związanego z regresją z uwagi na skalę i zasięg automatycznych testów regresji.

Z zastosowaniem metodyki DevOps wiążą się też jednak pewne ryzyka i wyzwania, wśród których można wymienić:

- konieczność zdefiniowania i ustanowienia potoku dostarczania DevOps;
- konieczność wprowadzenia i utrzymywania narzędzi do ciągłej integracji lub ciągłego dostarczania;
- konieczność przeznaczenia dodatkowych zasobów na automatyzację testów oraz trudności związane z wprowadzeniem i utrzymaniem mechanizmów automatyzacji.

Chociaż metodyka DevOps zakłada duży udział testów automatycznych, nadal konieczne jest również testowanie manualne — zwłaszcza z perspektywy użytkownika.

### 2.1.5. Przesunięcie w lewo (ang. *shift left approach*)

Zasada wczesnego testowania (patrz podrozdział 1.3) jest niekiedy nazywana „przesunięciem w lewo", ponieważ w myśl tej zasady testowanie odbywa się na wcześniejszym etapie cyklu wytwarzania oprogramowania. Przesunięcie w lewo sugeruje, że testowanie powinno odbywać się wcześniej (np. bez czekania na implementację kodu bądź integrację modułów), ale nie oznacza to, że należy zaniedbywać wykonywanie testów na dalszych etapach cyklu życia wytwarzania oprogramowania.

Poniżej przedstawiono kilka dobrych praktyk, które obrazują, w jaki sposób można wdrożyć podejście przesunięcie w lewo w testowaniu. Są to między innymi:

- dokonywanie przeglądu specyfikacji z punktu widzenia testowania (przegląd specyfikacji pozwala często znaleźć potencjalne defekty, takie jak niejasności, braki czy niespójności);
- pisanie przypadków testowych przed rozpoczęciem pisania kodu i uruchamianie kodu w jarzmie testowym podczas implementacji;
- korzystanie z mechanizmu ciągłej integracji (a jeszcze lepiej — ciągłego dostarczania), ponieważ mechanizm ten pozwala szybko uzyskiwać informacje zwrotne i wykonywać automatyczne testy modułowe w odniesieniu do kodu źródłowego przekazywanego do repozytorium kodu;
- przeprowadzanie analizy statycznej kodu źródłowego przed rozpoczęciem testowania dynamicznego lub w ramach zautomatyzowanego procesu;
- wykonywanie testowania niefunkcjonalnego już od poziomu testów modułowych wszędzie tam, gdzie jest to możliwe (jest to forma przesunięcia w lewo, ponieważ testy niefunkcjonalne są często wykonywane na późniejszych etapach cyklu wytwarzania oprogramowania, gdy jest już dostępny kompletny system wraz z reprezentatywnym środowiskiem testowym).

Przesunięcie w lewo może wymagać dodatkowych szkoleń, nakładów pracy i/lub kosztów na wcześniejszym etapie procesu, ale z założenia powinno to zostać zrekompensowane przez zmniejszenie nakładów pracy lub obniżenie kosztów na późniejszych etapach.

W przypadku przesunięcia w lewo ważne jest przekonanie interesariuszy o zasadności takiego podejścia i uzyskanie ich poparcia w tym zakresie.

### 2.1.6. Retrospektywy i doskonalenie procesów

Retrospektywy (zwane także spotkaniami poprojektowymi lub retrospektywami projektu) są często organizowane po zakończeniu projektu lub iteracji bądź osiągnięciu kamienia milowego związanego z przekazaniem do eksploatacji, ale w razie potrzeby mogą również odbywać się w innych momentach. Termin i przebieg retrospektywy zależy od przyjętego modelu cyklu wytwarzania oprogramowania. Podczas tego rodzaju spotkań uczestnicy (nie tylko testerzy, ale również np. programiści, architekci, właściciel produktu czy analitycy biznesowi) omawiają następujące kwestie:

- Jakie elementy zrealizowano pomyślnie i co należy zachować?
- Jakie działania zakończyły się niepowodzeniem i mogą zostać udoskonalone?
- W jaki sposób należy w przyszłości uwzględnić powyższe udoskonalenia i wykorzystać pomyślnie zrealizowane elementy?

Rezultaty należy udokumentować — zwykle robi się to w ramach sumarycznego raportu z testów (patrz sekcja 5.3.2). Retrospektywy mają kluczowe znaczenie dla pomyślnej realizacji zasady ciągłego

doskonalenia, dlatego ważne jest zweryfikowanie, czy zalecane usprawnienia zostały wprowadzone w życie.

Typowe korzyści z punktu widzenia testowania to między innymi:

- zwiększenie skuteczności i efektywności testów (np. poprzez wprowadzenie w życie sugestii dotyczących doskonalenia procesów);
- podniesienie jakości testaliów (np. w wyniku wspólnego przeglądu procesów testowych);
- zacieśnienie więzi w zespole i wspólne uczenie się (np. dzięki możliwości zgłaszania problemów i proponowania usprawnień);
- podniesienie jakości podstawy testów (np. dzięki możliwości identyfikowania i usuwania niedociągnięć związanych z jakością i zakresem wymagań);
- usprawnienie współpracy między programistami a testerami (np. dzięki regularnemu weryfikowaniu i optymalizowaniu zasad współpracy).

## 2.2. Poziomy testów i typy testów

Poziomy testów to grupy czynności testowych, które organizuje się i którymi zarządza się wspólnie. Każdy poziom testów jest instancją procesu testowego wykonywaną w odniesieniu do oprogramowania na danym etapie wytwarzania — od pojedynczych modułów po kompletne systemy lub, jeśli ma to zastosowanie w danym przypadku, po systemy systemów.

Poziomy testów są powiązane z innymi czynnościami wykonywanymi w ramach cyklu wytwarzania oprogramowania. W sekwencyjnych modelach cyklu wytwarzania oprogramowania poziomy testów często definiuje się w taki sposób, aby kryteria wyjścia jednego poziomu były elementem kryteriów wejścia kolejnego poziomu, natomiast w niektórych modelach iteracyjnych zasada ta może nie mieć zastosowania. Czynności związane z wytwarzaniem oprogramowania mogą obejmować wiele poziomów testów, a poziomy testów mogą zachodzić na siebie w czasie.

Typy testów to grupy czynności testowych związanych z konkretnymi charakterystykami jakościowymi, przy czym większość z tych czynności można wykonywać na każdym poziomie testów.

### 2.2.1. Poziomy testów

W niniejszym sylabusie opisano pięć poziomów testów.

- **Testowanie modułowe** (zwane także testowaniem jednostkowym lub testowaniem komponentów) skupia się na oddzielnym testowaniu poszczególnych modułów. Często wymaga ono stosowania określonych elementów pomocniczych, takich jak jarzma testowe lub struktury do testów jednostkowych (ang. *frameworks*). Testowanie modułowe jest zwykle wykonywane przez programistów w środowiskach tworzenia oprogramowania.
- **Testowanie integracji modułów** (zwane także testowaniem połączenia) skupia się na interfejsach i interakcjach między modułami. Sposób testowania integracji modułów zależy w dużej mierze od strategii integracji — może to być na przykład strategia zstępująca, strategia wstępująca bądź strategia typu „wielki wybuch” (ang. *big bang*).

- **Testowanie systemowe** skupia się na ogólnym zachowaniu i możliwościach całego systemu lub produktu. Często obejmuje również kompleksowe testowanie funkcjonalne wszystkich zadań, jakie system ten może wykonywać, oraz testowanie niefunkcjonalne charakterystyk jakościowych. W przypadku niektórych niefunkcjonalnych charakterystyk jakościowych wskazane jest przeprowadzanie testów kompletnego systemu w reprezentatywnym środowisku testowym (np. testowanie użyteczności), ale można również zastosować symulacje podsystemów. Testowanie systemowe może być wykonywane przez niezależny zespół testowy i jest powiązane ze specyfikacjami systemu.
- **Testowanie integracji systemów** skupia się na interfejsach łączących system podlegający testowaniu z innymi systemami oraz usługami zewnętrznymi. Do testowania integracji systemów niezbędne są odpowiednie środowiska testowe — w miarę możliwości zbliżone do środowiska produkcyjnego.
- **Testowanie akceptacyjne** skupia się na przeprowadzeniu walidacji i wykazaniu, że system jest gotowy do wdrożenia (tzn. zaspokaja potrzeby biznesowe użytkownika). W idealnych warunkach testowanie akceptacyjne powinni przeprowadzać docelowi użytkownicy. Najważniejsze formy testowania akceptacyjnego to: testowanie akceptacyjne przez użytkownika, operacyjne testy akceptacyjne, testowanie akceptacyjne zgodności z umową i testowanie akceptacyjne zgodności z prawem, testowanie alfa oraz testowanie beta.

Podstawą rozróżnienia między poszczególnymi poziomami testów (w celu uniknięcia nakładania się czynności testowych) są między innymi następujące atrybuty:

- przedmiot testów;
- cele testów;
- podstawa testów;
- defekty i awarie;
- podejście i odpowiedzialności.

### 2.2.2. Typy testów

Istnieje wiele typów testów, które można stosować w ramach projektów. W niniejszym sylabusie omówiono cztery z nich.

**Testowanie funkcjonalne** polega na dokonaniu oceny funkcji, które powinien realizować dany moduł lub system. Funkcje opisują to, „co” powinien robić dany przedmiot testów. Głównym celem testowania funkcjonalnego jest sprawdzenie kompletności funkcjonalnej, poprawności funkcjonalnej oraz adekwatności funkcjonalnej.

**Testowanie niefunkcjonalne** ma na celu dokonanie oceny atrybutów innych niż charakterystyki funkcjonalne modułu lub systemu. Testowanie niefunkcjonalne pozwala sprawdzić to, „jak dobrze” zachowuje się dany system. Głównym celem testowania niefunkcjonalnego jest sprawdzenie niefunkcjonalnych charakterystyk jakościowych oprogramowania.

Standard ISO/IEC 25010 podaje następującą klasyfikację niefunkcjonalnych charakterystyk jakościowych oprogramowania:

- wydajność;
- kompatybilność;
- użyteczność;
- niezawodność;
- zabezpieczenia;
- utrzymywalność;
- przenaszalność.

W pewnych przypadkach wskazane jest rozpoczęcie testowania niefunkcjonalnego na wczesnym etapie cyklu wytwarzania (np. w ramach przeglądów oraz testowania modułowego lub systemowego). Podstawą wielu testów niefunkcjonalnych są testy funkcjonalne — w praktyce wykorzystywane są te same testy funkcjonalne, ale celem jest sprawdzenie, czy podczas wykonywania danej funkcji spełniane są wymogi niefunkcjonalne (np. czy funkcja jest wykonywana w określonym czasie lub czy można ją przenieść na nową platformę). Zbyt późne wykrycie defektów odnoszących się do charakterystyk niefunkcjonalnych może być bardzo dużym zagrożeniem dla powodzenia projektu. Do wykonania testowania niefunkcjonalnego może być niekiedy niezbędne bardzo konkretne środowisko testowe, takie jak laboratorium użyteczności w przypadku testowania użyteczności.

**Testowanie czarnoskrzynkowe** (patrz podrozdział 4.2) opiera się na specyfikacjach, a testy wyprowadza się na podstawie dokumentacji zewnętrznej wobec przedmiotu testów. Głównym celem testowania czarnoskrzynkowego jest sprawdzenie zachowania systemu pod kątem zgodności ze specyfikacją.

**Testowanie białoskrzynkowe** (patrz podrozdział 4.3) ma charakter strukturalny, a testy wyprowadza się na podstawie implementacji lub struktury wewnętrznej danego systemu (np. kodu, architektury, przepływów pracy i przepływów danych). Głównym celem testowania białoskrzynkowego jest uzyskanie akceptowalnego poziomu pokrycia testowego bazowej struktury systemu.

Wszystkie cztery typy testów wymienione powyżej można stosować na wszystkich poziomach testów, chociaż działania podejmowane na każdym z tych poziomów będą inaczej ukierunkowane. Do wyprowadzania warunków testowych i przypadków testowych na potrzeby wszystkich wspomnianych typów testów można używać różnych technik testowania.

### 2.2.3. Testowanie potwierdzające i testowanie regresji

W modułach lub systemach są często wprowadzane zmiany mające na celu rozszerzenie funkcjonalności poprzez dodanie nowego elementu bądź przywrócenie prawidłowej funkcjonalności poprzez usunięcie defektu. Oznacza to, że proces testowania powinien również obejmować testowanie potwierdzające i testowanie regresji.

**Testowanie potwierdzające** ma na celu sprawdzenie, czy pierwotny defekt został pomyślnie usunięty. Zależnie od poziomu ryzyka skorygowaną wersję oprogramowania można przetestować na kilka sposobów, w tym poprzez:

- wykonanie wszystkich przypadków testowych, które wcześniej nie zostały zaliczone z powodu defektu lub

- dodanie nowych testów w celu pokrycia ewentualnych zmian, które były niezbędne do usunięcia defektu.

Jeśli jednak przy usuwaniu defektów brakuje czasu lub środków finansowych, testowanie potwierdzające można ograniczyć do wykonania kroków potrzebnych do odtworzenia awarii spowodowanej defektem i sprawdzenia, czy tym razem ona nie występuje.

**Testowanie regresji** pozwala sprawdzić, czy wprowadzona zmiana (w tym także poprawka, która była już przedmiotem testowania potwierdzającego) nie spowodowała negatywnych konsekwencji. Konsekwencje takie mogą dotyczyć modułu, w którym wprowadzono zmianę, innych modułów tego samego systemu, a nawet innych podłączonych systemów, w związku z czym testowanie regresji może wykraczać poza przedmiot testów i obejmować również środowisko, w którym się on znajduje. Dlatego zalecane jest przeprowadzenie w pierwszej kolejności analizy wpływu mającej na celu zoptymalizowanie zasięgu testowania regresji, co pozwoli wskazać elementy oprogramowania, na które będzie mogła wpłynąć wprowadzona zmiana.

Zestawy testów regresji są wykonywane wielokrotnie, a liczba związanych z nimi przypadków testowych rośnie z każdą iteracją lub każdą wersją, w związku z czym testowanie regresji świetnie nadaje się do automatyzacji. Dlatego też automatyzację tego rodzaju testów należy rozpocząć na początkowym etapie projektu. Automatyzacja testów regresji jest również dobrą praktyką w przypadku korzystania z mechanizmu ciągłej integracji — na przykład w ramach metodyki DevOps (patrz sekcja 2.1.4). Zależnie od sytuacji, automatyzacja ta może obejmować testy regresji na różnych poziomach testów.

Testowanie potwierdzające i/lub testowanie regresji należy wykonywać w odniesieniu do przedmiotu testów na wszystkich poziomach testów, na których zostały usunięte defekty i/lub wprowadzone zmiany.

## 2.3. Testowanie pielęgnacyjne

Wyróżnia się kilka kategorii testowania pielęgnacyjnego. Pielęgnacja może obejmować między innymi działania naprawcze, działania wynikające z konieczności dostosowania oprogramowania do zmian w środowisku oraz działania mające na celu zwiększenie wydajności lub utrzymywalności (szczegółowe informacje na ten temat zawiera standard ISO/IEC 14764). Tym samym testowanie pielęgnacyjne może wynikać z wdrożenia oprogramowania lub przekazania go do eksploatacji zarówno w sposób planowy, jak i niezaplanowany — na przykład w związku z poprawkami doraźnymi (ang. *hot fix*). Przed dokonaniem zmiany można przeprowadzić analizę wpływu, aby ustalić, czy zmianę tę należy faktycznie wprowadzić (z uwagi na potencjalne konsekwencje w innych obszarach systemu). Ponadto jeśli dany system znajduje się w eksploatacji, testowanie zmian obejmuje zarówno sprawdzenie, czy zmiana została wprowadzona pomyślnie, jak i wykrycie ewentualnych regresji w niezmienionych częściach systemu (czyli zwykle w większości jego obszarów).

Na zakres testowania pielęgnacyjnego wpływają zwykle:

- poziom ryzyka związanego ze zmianą;
- wielkość dotychczasowego systemu;
- wielkość wprowadzonej zmiany.

Zdarzenia wywołujące pielęgnację i testowanie pielęgnacyjne można podzielić na następujące kategorie:

- Modyfikacje. Ta kategoria obejmuje między innymi zaplanowane udoskonalenia (wprowadzane w postaci nowych wersji oprogramowania), zmiany korekcyjne i poprawki doraźne.

- Uaktualnienia lub migracje środowiska produkcyjnego. Ta kategoria obejmuje między innymi przejście z jednej platformy na inną, co może wiązać się z koniecznością przeprowadzenia testów związanych z nowym środowiskiem i zmienionym oprogramowaniem bądź testów konwersji danych (w przypadku migracji danych z innej aplikacji do pielęgnowanego systemu).
- Wycofanie. Ta kategoria dotyczy sytuacji, w której okres użytkowania aplikacji dobiega końca. W przypadku wycofywania systemu może być konieczne przetestowanie archiwizacji danych, jeśli zachodzi potrzeba ich przechowywania przez dłuższy czas. Ponadto jeśli w okresie archiwizacji będzie wymagany dostęp do niektórych danych, może być konieczne przetestowanie procedur przywracania i odtwarzania danych po archiwizacji.

# 3. Testowanie statyczne — 80 minut

**Słowa kluczowe**

analiza statyczna, anomalia, inspekcja, przegląd, przegląd formalny, przegląd nieformalny, przegląd techniczny, przejrzenie, testowanie dynamiczne, testowanie statyczne

**Cele nauczania w rozdziale 3:**

### 3.1 Podstawy testowania statycznego

FL-3.1.1 (K1) Kandydat rozpoznaje typy produktów, które mogą być badane przy użyciu poszczególnych technik testowania statycznego.

FL-3.1.2 (K2) Kandydat wyjaśnia korzyści wynikające z testowania statycznego.

FL-3.1.3 (K2) Kandydat porównuje i zestawia ze sobą testowanie statyczne i dynamiczne.

### 3.2 Informacje zwrotne i proces przeglądu

FL-3.2.1 (K1) Kandydat pamięta korzyści wynikające z wczesnego i częstego otrzymywania informacji zwrotnych od interesariuszy.

FL-3.2.2 (K2) Kandydat podsumowuje czynności wykonywane w ramach procesu przeglądu.

FL-3.2.3 (K1) Kandydat pamięta, jakie obowiązki są przypisane do najważniejszych ról w trakcie wykonywania przeglądów.

FL-3.2.4 (K2) Kandydat porównuje i zestawia ze sobą różne typy przeglądów.

FL-3.2.5 (K1) Kandydat pamięta, jakie czynniki decydują o powodzeniu przeglądu.

## 3.1. Podstawy testowania statycznego

W przeciwieństwie do testowania dynamicznego testowanie statyczne nie wymaga uruchamiania testowanego oprogramowania. W testowaniu statycznym oceny kodu, specyfikacji procesów, specyfikacji architektury systemu i innych produktów pracy dokonuje się poprzez ich manualne zbadanie (np. w ramach przeglądu) lub poprzez zastosowanie odpowiedniego narzędzia (np. w ramach analizy statycznej). Wśród celów testowania statycznego można wymienić podnoszenie jakości, wykrywanie defektów oraz ocenianie charakterystyk takich jak czytelność, kompletność, poprawność, testowalność czy spójność. Testowanie statyczne można stosować zarówno w przypadku weryfikacji, jak i w przypadku walidacji.

Testerzy, przedstawiciele jednostek biznesowych i programiści współpracują ze sobą podczas sesji mapowania przykładów (ang. *example mapping*), wspólnego pisania historyjek użytkownika i doprecyzowywania backlogu (ang. *backlog refinement sessions*), dbając o to, aby historyjki użytkownika i związane z nimi produkty pracy były zgodne z określonymi kryteriami — na przykład z definicją gotowości (ang. *Definition of Ready*, patrz sekcja 5.1.3). Mogą oni przy tym korzystać z technik przeglądu, aby zyskać pewność, że tworzone historyjki użytkownika są kompletne i zrozumiałe oraz zawierają testowalne kryteria akceptacji. Zadając właściwe pytania, testerzy mogą również badać, kwestionować i udoskonalać proponowane historyjki użytkownika.

Analiza statyczna pozwala rozpoznać problemy przed rozpoczęciem testowania dynamicznego, a przy tym jest często mniej pracochłonna, ponieważ nie wymaga tworzenia przypadków testowych i odbywa się zwykle z wykorzystaniem narzędzi (patrz rozdział 6). Często jest też elementem mechanizmów ciągłej integracji (patrz sekcja 2.1.4). Chociaż głównym zastosowaniem analizy statycznej jest wykrywanie konkretnych defektów kodu, metoda ta może również służyć do oceny utrzymywalności i poziomu zabezpieczenia systemu. Przykładami narzędzi do analizy statycznej są również narzędzia do sprawdzania pisowni i czytelności.

### 3.1.1. Produkty pracy badane metodą testowania statycznego

Przy użyciu technik testowania statycznego można zbadać niemal wszystkie produkty pracy, na przykład: dokumenty zawierające specyfikacje wymagań, kod źródłowy, plany testów, przypadki testowe, pozycje backlogu (ang. *product backlog items*), karty opisu testów, dokumentację projektu, umowy oraz modele.

Należy jednak pamiętać, że o ile przedmiotem przeglądu może być dowolny produkt pracy, który da się przeczytać i zrozumieć, o tyle w przypadku analizy statycznej niezbędna jest struktura (np. modele, kod lub tekst z formalną składnią), względem której można sprawdzić badane produkty pracy.

Do produktów pracy, które nie nadają się do objęcia testowaniem statycznym, należą między innymi produkty trudne do zinterpretowania przez człowieka oraz produkty, których nie należy analizować za pomocą narzędzi (np. kod wykonywalny innych firm, którego nie wolno badać ze względów prawnych).

### 3.1.2. Korzyści wynikające z testowania statycznego

Testowanie statyczne pozwala wykryć defekty w najwcześniejszych fazach cyklu wytwarzania oprogramowania, a tym samym realizuje zasadę wczesnego testowania (patrz podrozdział 1.3). Ponadto metoda ta umożliwia identyfikowanie defektów, których nie da się wykryć podczas testowania dynamicznego — takich jak nieosiągalny kod, niewłaściwie zaimplementowane wzorce projektowe czy defekty w niewykonywalnych produktach pracy.

Testowanie statyczne umożliwia dokonywanie oceny jakości produktów pracy i budowanie zaufania do nich. Weryfikując udokumentowane wymagania, interesariusze mogą upewnić się, że opisują one ich rzeczywiste potrzeby. Ponadto — z uwagi na fakt, że testowanie statyczne można wykonywać we wczesnych fazach cyklu wytwarzania oprogramowania — zaangażowani w ten proces interesariusze mogą wypracować wspólny punkt widzenia. Inną ważną korzyścią jest usprawnienie wymiany informacji pomiędzy interesariuszami. Z tego powodu zaleca się, aby w proces testowania statycznego zaangażowane było możliwie szerokie grono zainteresowanych osób.

Chociaż przeprowadzanie przeglądów może być kosztowne, łączne koszty projektu są zwykle dużo niższe niż w przypadku rezygnacji z tego procesu, ponieważ dzięki przeglądom zmniejsza się czasochłonność i pracochłonność usuwania defektów na późniejszych etapach projektu.

Analiza statyczna pozwala wykrywać defekty kodu bardziej efektywnie niż testowanie dynamiczne, co przekłada się zwykle na zmniejszenie liczby tego rodzaju defektów oraz obniżenie łącznych nakładów pracy związanych z wytwarzaniem oprogramowania.

### 3.1.3. Różnice między testowaniem statycznym a dynamicznym

Testowanie statyczne i testowanie dynamiczne wzajemnie się uzupełniają. Techniki te mają podobne cele, takie jak wspomaganie wykrywania defektów w produktach pracy (patrz sekcja 1.1.1), ale różnią się pod pewnymi względami, które opisano poniżej.

- Zarówno testowanie statyczne, jak i testowanie dynamiczne (z analizą awarii) może prowadzić do wykrycia defektów, jednak istnieją pewne rodzaje defektów, które można wykryć tylko jedną z powyższych metod.
- Testowanie statyczne umożliwia bezpośrednie wykrywanie defektów, natomiast testowanie dynamiczne powoduje występowanie awarii, które są następnie analizowane w celu zidentyfikowania związanych z nimi defektów.
- Testowanie statyczne pozwala łatwiej wykryć defekty, które znajdują się na rzadko wykonywanych ścieżkach w kodzie lub w miejscach trudno dostępnych podczas testowania dynamicznego.
- Testowanie statyczne można stosować do niewykonywalnych produktów pracy, a testowanie dynamiczne — tylko do produktów wykonywalnych.
- Testowanie statyczne może służyć do mierzenia charakterystyk jakościowych, które nie są zależne od wykonywania kodu (takich jak utrzymywalność), a testowanie dynamiczne — do mierzenia charakterystyk jakościowych zależnych od wykonywania kodu (takich jak wydajność).

Przykładami typowych defektów, które są łatwiejsze i/lub tańsze do wykrycia przy zastosowaniu metody testowania statycznego, są między innymi:

- defekty w wymaganiach (np. niespójności, niejednoznaczności, sprzeczności, przeoczenia, nieścisłości czy powtórzenia);
- defekty w projekcie (np. nieefektywne struktury baz danych bądź niewłaściwa modularyzacja);
- niektóre typy defektów w kodzie (np. zmienne z niezdefiniowanymi wartościami, niezadeklarowane zmienne, nieosiągalny lub wielokrotnie powtórzony kod bądź kod o nadmiernej złożoności);

- odchylenia od standardów (np. brak zgodności z konwencjami nazewnictwa określonymi w standardach tworzenia kodu);
- niepoprawne specyfikacje interfejsów (np. niezgodność liczby, typu lub kolejności parametrów);
- określone rodzaje słabych punktów zabezpieczeń (np. podatność na przepełnienie bufora);
- luki lub nieścisłości w pokryciu podstawy testów (np. brak testów odpowiadających kryteriom akceptacji).

## 3.2. Informacje zwrotne i proces przeglądu

### 3.2.1. Korzyści wynikające z wczesnego i częstego otrzymywania informacji zwrotnych od interesariuszy

Przekazywane w odpowiednim czasie i z odpowiednią częstotliwością informacje zwrotne umożliwiają wczesne rozpoznawanie i sygnalizowanie potencjalnych problemów z jakością. Jeśli zaangażowanie interesariuszy w trakcie cyklu wytwarzania oprogramowania jest niewielkie, wytwarzany produkt może okazać się niezgodny z ich pierwotną lub obecną wizją. Niespełnienie oczekiwań interesariuszy może mieć poważne konsekwencje, takie jak konieczność wprowadzenia kosztownych poprawek, niedotrzymanie terminów, przerzucanie się odpowiedzialnością, a nawet niepowodzenie całego projektu.

Częste przekazywanie przez interesariuszy informacji zwrotnych we wszystkich fazach cyklu wytwarzania oprogramowania pozwala zapobiec ewentualnym nieporozumieniom w kwestii wymagań, a także umożliwia szybsze analizowanie i wprowadzanie ewentualnych zmian. Dzięki temu zespół tworzący oprogramowanie dysponuje lepszą wiedzą na temat budowanego systemu, przez co może skoncentrować się na działaniach, które przynoszą interesariuszom największe korzyści i pozwalają najskuteczniej łagodzić zidentyfikowane ryzyka.

### 3.2.2. Czynności wykonywane w procesie przeglądu

W standardzie ISO/IEC 20246 zdefiniowano ogólny proces przeglądu, który wyznacza usystematyzowane, a zarazem elastyczne ramy będące podstawą do wypracowania szczegółowego procesu dostosowanego do konkretnej sytuacji. Jeśli wymagany przegląd ma charakter bardziej formalny, konieczne jest wykonanie większej liczby zadań opisanych w kontekście poszczególnych czynności.

Wiele produktów pracy ma zbyt duże rozmiary, aby można je było objąć pojedynczym przeglądem. W takiej sytuacji w celu przeanalizowania całego produktu pracy proces przeglądu można przeprowadzić kilkakrotnie.

W procesie przeglądu można wyróżnić następujące czynności:

- **Planowanie.** W fazie planowania określa się zakres przeglądu, w tym cel przeglądu, produkt pracy będący jego przedmiotem, oceniane charakterystyki jakościowe, obszary wymagające szczególnej uwagi, kryteria wyjścia, informacje pomocnicze (takie jak standardy), nakład pracy oraz ramy czasowe.
- **Rozpoczęcie przeglądu.** Na etapie rozpoczęcia przeglądu należy upewnić się, że wszystkie zaangażowane osoby i wszystkie niezbędne elementy są gotowe do jego przeprowadzenia. Należy między innymi sprawdzić, czy każdy uczestnik ma dostęp do produktu pracy będącego

przedmiotem przeglądu, zna swoją rolę i swoje obowiązki oraz otrzymał wszystkie materiały niezbędne do przeprowadzenia przeglądu.

- **Przegląd indywidualny.** Każdy przeglądający dokonuje przeglądu indywidualnego, aby ocenić jakość produktu pracy będącego przedmiotem przeglądu oraz zidentyfikować ewentualne anomalie, zalecenia i pytania, korzystając w tym celu z jednej lub kilku technik przeglądu (takich jak przegląd oparty na liście kontrolnej czy przegląd oparty na scenariuszach). Dokładniejsze informacje na temat poszczególnych technik przeglądu zawiera standard ISO/IEC 20246. Przeglądający odnotowują wszystkie zidentyfikowane przez siebie anomalie, zalecenia i pytania.
- **Przekazanie informacji i analiza.** Z uwagi na to, że anomalie zidentyfikowane w trakcie przeglądu nie muszą być defektami, konieczne jest przeanalizowanie i omówienie każdej z nich, a następnie określenie jej statusu, wyznaczenie osoby odpowiedzialnej i wskazanie wymaganych działań. Zwykle odbywa się to w ramach spotkania związanego z przeglądem, podczas którego uczestnicy podejmują również decyzję co do poziomu jakości produktu pracy będącego przedmiotem przeglądu oraz wymaganych dalszych działań. Może się również okazać, że zakończenie podjętych działań będzie wymagało zorganizowania kolejnego przeglądu.
- **Usunięcie defektów i raportowanie.** W odniesieniu do każdego defektu należy sporządzić raport o defekcie, aby umożliwić zweryfikowanie wykonania działań naprawczych. Po spełnieniu kryteriów wyjścia można dokonać odbioru produktu pracy. Wyniki przeglądu należy ująć w raporcie.

### 3.2.3. Role i obowiązki w przeglądach

W przeglądach uczestniczą różni interesariusze, którzy mogą pełnić kilka ról. Poniżej omówiono najważniejsze role i związane z nimi odpowiedzialności:

- Kierownik — decyduje o tym, co ma być przedmiotem przeglądu, a także udostępnia niezbędne zasoby — w tym wyznacza pracowników oraz określa ramy czasowe przeglądu.
- Autor — tworzy produkt pracy będący przedmiotem przeglądu i usuwa występujące w nim defekty.
- Moderator (zwany także facylitatorem) — dba o sprawny przebieg spotkań związanych z przeglądem. Występuje w roli mediatora, zarządza czasem oraz zapewnia bezpieczne warunki, w których każdy uczestnik przeglądu może swobodnie wyrażać swoje zdanie.
- Protokolant (zwany także rejestrującym) — gromadzi informacje o anomaliach przekazane przez przeglądających i protokołuje informacje związane z przeglądem, w tym informacje o podjętych decyzjach oraz o nowych anomaliach stwierdzonych w trakcie spotkania związanego z przeglądem.
- Przeglądający — wykonuje przegląd. Rolę tę może pełnić osoba pracująca przy projekcie, ekspert merytoryczny lub dowolny inny interesariusz.
- Lider przeglądu — ponosi ogólną odpowiedzialność za przegląd, w tym decyduje o tym, kto ma wziąć udział w przeglądzie, oraz określa miejsce i termin przeglądu.

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

### 3.2.4. Typy przeglądów

Istnieje wiele typów przeglądów — od nieformalnych po formalne. Wymagany stopień sformalizowania przeglądu zależy od takich czynników jak: przyjęty model cyklu wytwarzania oprogramowania, dojrzałość procesu wytwarzania oprogramowania, krytyczność i złożoność produktu pracy będącego przedmiotem przeglądu, wymogi prawne czy konieczność prowadzenia ścieżki audytu. Ten sam produkt pracy może być objęty różnymi typami przeglądów, na przykład najpierw przeglądem nieformalnym, a następnie przeglądem bardziej sformalizowanym.

Wybór właściwego typu przeglądu ma zasadnicze znaczenie dla osiągnięcia wymaganych celów przeglądu (patrz sekcja 3.2.5). Wyboru tego dokonuje się nie tylko na podstawie celów, ale również na podstawie czynników takich jak: potrzeby projektu, dostępne zasoby, typ produktu pracy i związane z nim ryzyka, dziedzina biznesowa oraz kultura organizacyjna firmy.

Poniżej omówiono kilka często stosowanych typów przeglądów:

- **Przegląd nieformalny.** Przeglądy nieformalne nie przebiegają zgodnie ze zdefiniowanym procesem, a uzyskanych dzięki nim informacji nie trzeba formalnie dokumentować. Głównym celem jest wykrycie anomalii.
- **Przejrzenie.** Przejrzenie, które prowadzi autor, może służyć wielu celom, takim jak: dokonanie oceny jakości produktu pracy i zwiększenie zaufania do niego, edukowanie przeglądających, osiągnięcie konsensusu, wygenerowanie nowych pomysłów, zmotywowanie autorów do udoskonalania przyszłych produktów pracy i stworzenie im warunków do tego oraz wykrycie anomalii. Przed rozpoczęciem przejrzenia przeglądający mogą przeprowadzić przegląd indywidualny, ale nie jest to konieczne.
- **Przegląd techniczny.** Przegląd techniczny wykonują przeglądający, którzy dysponują odpowiednimi kwalifikacjami technicznymi, a nad przebiegiem procesu czuwa moderator. Celem przeglądu technicznego jest nie tylko osiągnięcie konsensusu i podjęcie decyzji w sprawie problemu technicznego, ale również wykrycie anomalii, dokonanie oceny jakości produktu pracy i zwiększenie zaufania do niego, wygenerowanie nowych pomysłów oraz zmotywowanie autorów do wprowadzania udoskonaleń i stworzenie im warunków do tego.
- **Inspekcja.** Inspekcja jest najbardziej formalnym typem przeglądu, w związku z czym odbywa się zgodnie z pełnym ogólnym procesem przeglądu (patrz sekcja 3.2.2). Głównym celem jest wykrycie jak największej liczby anomalii, a pozostałe cele to: dokonanie oceny jakości produktu pracy i zwiększenie zaufania do niego oraz zmotywowanie autorów do wprowadzania udoskonaleń i stworzenie im warunków do tego. Zbierane są metryki wykorzystywane następnie do udoskonalania cyklu wytwarzania oprogramowania (w tym procesu inspekcji). W ramach inspekcji autor nie może być liderem przeglądu ani protokolantem.

### 3.2.5. Czynniki powodzenia związane z przeglądami

Można wyróżnić kilka czynników, które decydują o pomyślnym przebiegu przeglądu, w tym:

- określenie jednoznacznych celów i mierzalnych kryteriów wyjścia (przy czym celem nie powinna być w żadnym razie ocena uczestników);
- wybór odpowiedniego typu przeglądu, który pozwoli osiągnąć zakładane cele oraz będzie odpowiedni do typu produktu pracy, uczestników przeglądu, potrzeb projektu i kontekstu;

- przeprowadzanie przeglądów w odniesieniu do mniejszych partii materiału, tak aby przeglądający nie tracili koncentracji podczas przeglądu indywidualnego i/lub spotkania związanego z przeglądem (jeśli jest organizowane);
- przekazywanie informacji zwrotnych z przeglądów interesariuszom i autorom, tak aby mogli oni udoskonalać produkt i usprawniać swoje działania (patrz sekcja 3.2.1);
- wyznaczenie uczestnikom wystarczającej ilości czasu na przygotowanie się do przeglądu;
- uzyskanie wsparcia kierownictwa dla procesu przeglądu;
- włączenie przeglądów w kulturę organizacyjną w celu stworzenia atmosfery sprzyjającej poszerzaniu wiedzy i doskonaleniu procesów;
- zapewnienie wszystkim uczestnikom należytego przeszkolenia niezbędnego do prawidłowego wykonywania wyznaczonych im ról;
- dbanie o sprawny przebieg spotkań.

# 4. Analiza i projektowanie testów — 390 minut

**Słowa kluczowe**

analiza wartości brzegowych, białoskrzynkowa technika testowania, czarnoskrzynkowa technika testowania, element pokrycia, kryteria akceptacji, podejście do testowania oparte na współpracy, podział na klasy równoważności, pokrycie, pokrycie gałęzi, pokrycie instrukcji kodu, technika testowania oparta na doświadczeniu, technika testowania, testowanie eksploracyjne, testowanie przejść pomiędzy stanami, testowanie w oparciu o tablicę decyzyjną, testowanie w oparciu o listę kontrolną, wytwarzanie sterowane testami akceptacyjnymi, zgadywanie błędów

**Cele nauczania w rozdziale 4:**

### 4.1 Ogólna charakterystyka technik testowania

FL-4.1.1 (K2) Kandydat rozróżnia czarnoskrzynkowe i białoskrzynkowe techniki testowania oraz techniki testowania oparte na doświadczeniu.

### 4.2 Czarnoskrzynkowe techniki testowania

FL-4.2.1 (K3) Kandydat używa techniki podziału na klasy równoważności, aby zaprojektować przypadki testowe.

FL-4.2.2 (K3) Kandydat używa techniki analizy wartości brzegowych, aby zaprojektować przypadki testowe.

FL-4.2.3 (K3) Kandydat używa techniki testowania w oparciu o tablicę decyzyjną, aby zaprojektować przypadki testowe.

FL-4.2.4 (K3) Kandydat używa techniki testowania przejść pomiędzy stanami, aby zaprojektować przypadki testowe.

### 4.3 Białoskrzynkowe techniki testowania

FL-4.3.1 (K2) Kandydat wyjaśnia pojęcie testowanie instrukcji.

FL-4.3.2 (K2) Kandydat wyjaśnia pojęcie testowanie gałęzi.

FL-4.3.3 (K2) Kandydat wyjaśnia korzyści wynikające z testowania białoskrzynkowego.

### 4.4 Techniki testowania oparte na doświadczeniu

FL-4.4.1 (K2) Kandydat wyjaśnia pojęcie zgadywanie błędów.

FL-4.4.2 (K2) Kandydat wyjaśnia pojęcie testowanie eksploracyjne.

FL-4.4.3 (K2) Kandydat wyjaśnia pojęcie testowanie w oparciu o listę kontrolną.

### 4.5 Podejścia do testowania oparte na współpracy

FL-4.5.1 (K2) Kandydat wyjaśnia, w jaki sposób należy pisać historyjki użytkownika we współpracy z programistami i przedstawicielami jednostek biznesowych.

FL-4.5.2 (K2) Kandydat klasyfikuje różne sposoby pisania kryteriów akceptacji.

FL-4.5.3 (K3) Kandydat używa metody wytwarzania sterowanego testami akceptacyjnymi (ATDD), aby zaprojektować przypadki testowe.

## 4.1. Ogólna charakterystyka technik testowania

Techniki testowania pomagają testerom w przeprowadzaniu analizy testów (która odpowiada na pytanie „co należy przetestować") oraz w projektowaniu testów (które odpowiada na pytanie „jak należy testować"). Za pomocą tego typu technik można systematycznie opracować stosunkowo niewielki, ale wystarczający zbiór przypadków testowych. Ponadto techniki testowania ułatwiają definiowanie warunków testowych oraz identyfikowanie elementów pokrycia i danych testowych na etapie analizy i projektowania testów. Więcej informacji na temat technik testowania i odpowiadających im miar można znaleźć w standardzie ISO/IEC/IEEE 29119-4 oraz w (Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jorgensen 2014, Ammann 2016, Forgács 2019).

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

**Czarnoskrzynkowe techniki testowania** (zwane także technikami opartymi na specyfikacji) bazują na analizie wyspecyfikowanego zachowania przedmiotu testów bez odwoływania się do jego struktury wewnętrznej. Oznacza to, że przypadki testowe są niezależne od sposobu implementacji oprogramowania, a co za tym idzie mogą być nadal stosowane w przypadku zmiany implementacji, która nie pociąga za sobą zmiany wymaganego zachowania.

Podstawą **białoskrzynkowych technik testowania** (zwanych także technikami opartymi na strukturze) jest analiza struktury wewnętrznej przedmiotu testów i wykonywanego w nim przetwarzania. Przypadki testowe są uzależnione od sposobu, w jaki zaprojektowano dane oprogramowanie, w związku z czym można je tworzyć dopiero po zaprojektowaniu lub zaimplementowaniu przedmiotu testów.

**Techniki testowania oparte na doświadczeniu** pozwalają wykorzystać wiedzę i doświadczenie testerów do projektowania i implementowania przypadków testowych. Skuteczność tego rodzaju technik zależy w dużej mierze od umiejętności testera. Techniki oparte na doświadczeniu pozwalają wykrywać defekty, które łatwo jest przeoczyć w przypadku stosowania technik czarnoskrzynkowych i białoskrzynkowych, a tym samym znakomicie uzupełniają powyższe techniki.

## 4.2. Czarnoskrzynkowe techniki testowania

W kolejnych sekcjach omówione zostały następujące powszechnie stosowane czarnoskrzynkowe techniki testowania:

- podział na klasy równoważności;
- analiza wartości brzegowych;
- testowanie w oparciu o tablicę decyzyjną;
- testowanie przejść pomiędzy stanami.

### 4.2.1. Podział na klasy równoważności

Technika podziału na klasy równoważności polega na dzieleniu danych na klasy (zwane klasami równoważności) zgodnie z założeniem, że każda klasa będzie zawierała elementy, które mają być przetwarzane przez przedmiot testów w ten sam sposób. U podstaw tej techniki leży teoria, zgodnie z którą jeśli przypadek testowy służący do testowania jednej wartości z klasy równoważności wykryje defekt, to defekt ten powinien również zostać wykryty przez przypadki testowe służące do testowania każdej innej

wartości należącej do tej samej klasy. Oznacza to, że wystarczające jest utworzenie jednego testu w odniesieniu do każdej z klas.

Klasy równoważności można wyznaczać w odniesieniu do wszelkich elementów danych, które są związane z przedmiotem testów, takich jak: dane wejściowe, dane wyjściowe, elementy konfiguracji, wartości wewnętrzne, wartości zależne od czasu oraz parametry interfejsu. Klasy mogą być ciągłe lub dyskretne, uporządkowane lub nieuporządkowane oraz skończone lub nieskończone, nie mogą natomiast nakładać się na siebie ani być pustymi zbiorami.

W przypadku prostych przedmiotów testów podział na klasy równoważności nie sprawia zwykle trudności, ale w praktyce zrozumienie tego, jak dany obiekt testowy będzie traktować poszczególne wartości, bywa skomplikowane. Z tego powodu przy dokonywaniu podziału należy zachować ostrożność.

Klasa równoważności zawierająca poprawne wartości jest nazywana poprawną klasą równoważności, a klasa zawierająca wartości niepoprawne — niepoprawną klasą równoważności. Definicje wartości poprawnych i niepoprawnych mogą przy tym różnić się w zależności od zespołu lub organizacji. Za wartości poprawne mogą być na przykład uważane takie, które powinny być przetwarzane przez przedmiot testów, lub takie, w przypadku których sposób przetwarzania określa specyfikacja. Z kolei za wartości niepoprawne mogą być uznawane wartości, które przedmiot testów powinien zignorować lub odrzucić, bądź wartości, w odniesieniu do których w specyfikacji przedmiotu testów nie zdefiniowano sposobu przetwarzania.

W przypadku podziału na klasy równoważności elementami pokrycia są klasy równoważności. Warunkiem uzyskania stuprocentowego pokrycia przy korzystaniu z tej techniki jest sprawdzenie za pomocą przypadków testowych wszystkich zidentyfikowanych klas (w tym klas niepoprawnych) poprzez pokrycie każdej klasy co najmniej raz. Pokrycie mierzy się jako iloraz liczby klas sprawdzonych za pomocą co najmniej jednego przypadku testowego przez łączną liczbę zidentyfikowanych klas, a uzyskaną wartość wyraża się w procentach.

W wielu przedmiotach testów występuje po kilka zbiorów klas (przykładem mogą być przedmioty testów mające więcej niż jeden parametr wejściowy), co oznacza, że przypadek testowy będzie pokrywać klasy należące do kilku różnych zbiorów. Najprostszym kryterium pokrycia w przypadku występowania wielu zbiorów klas jest pokrycie typu „każdy wybór” (ang. *each choice*) (Ammann 2016). Wymaga ono, aby przypadki testowe sprawdzały każdą klasę z każdego zbioru klas co najmniej raz. Pokrycie to nie uwzględnia kombinacji klas.

### 4.2.2. Analiza wartości brzegowych

Technika analizy wartości brzegowych polega na sprawdzaniu wartości brzegowych klas równoważności, w związku z czym może być stosowana tylko w odniesieniu do klas uporządkowanych. Wartościami brzegowymi klasy równoważności są jej wartość minimalna i maksymalna. W przypadku analizy wartości brzegowych zakłada się, że jeśli dwa elementy należą do tej samej klasy, to wszystkie elementy leżące pomiędzy nimi również muszą należeć do tej klasy.

Opisywana technika skupia się na wartościach brzegowych klas ze względu na większe prawdopodobieństwo popełnienia przez programistów pomyłek właśnie w przypadku takich wartości. Typowe defekty wykrywane metodą analizy wartości brzegowych znajdują się tam, gdzie zaimplementowane wartości brzegowe zostały omyłkowo umieszczone powyżej lub poniżej zamierzonego położenia lub całkowicie pominięte.

W niniejszym sylabusie omówiono dwa warianty analizy wartości brzegowych: analizę dwupunktową i analizę trójpunktową. Warianty te różnią się liczbą elementów pokrycia przypadających na każdą z wartości brzegowych, które muszą zostać sprawdzone w celu uzyskania stuprocentowego pokrycia.

W przypadku dwupunktowej analizy wartości brzegowych (Craig 2002, Myers 2011) na każdą wartość brzegową przypadają dwa elementy pokrycia: sama wartość brzegowa oraz najbliższa jej wartość należąca do sąsiedniej klasy równoważności. Warunkiem uzyskania stuprocentowego pokrycia przy korzystaniu z dwupunktowej analizy wartości brzegowych jest sprawdzenie za pomocą przypadków testowych wszystkich elementów pokrycia, czyli wszystkich zidentyfikowanych wartości brzegowych. Pokrycie mierzy się jako iloraz liczby sprawdzonych wartości brzegowych przez łączną liczbę zidentyfikowanych wartości brzegowych, a uzyskaną wartość wyraża się w procentach.

W przypadku trójpunktowej analizy wartości brzegowych (Koomen 2006, O'Regan 2019) na każdą wartość brzegową przypadają trzy elementy pokrycia: sama wartość brzegowa oraz obie wartości sąsiednie. W związku z tym w ramach trójpunktowej analizy wartości brzegowych niektóre elementy pokrycia mogą nie być wartościami brzegowymi. Warunkiem uzyskania stuprocentowego pokrycia przy korzystaniu z trójpunktowej analizy wartości brzegowych jest sprawdzenie za pomocą przypadków testowych wszystkich elementów pokrycia, czyli zidentyfikowanych wartości brzegowych i ich wartości sąsiednich. Pokrycie mierzy się jako iloraz liczby sprawdzonych wartości brzegowych i wartości sąsiednich przez łączną liczbę zidentyfikowanych wartości brzegowych i ich wartości sąsiednich, a uzyskaną wartość wyraża się w procentach.

Trójpunktowa analiza wartości brzegowych jest bardziej rygorystyczna niż analiza dwupunktowa, ponieważ pozwala wykryć defekty, które można przeoczyć w przypadku tej ostatniej. Na przykład jeśli decyzja „if (x ≤ 10) …" zostanie błędnie zaimplementowana jako „if (x = 10) …", dane testowe uzyskane w ramach dwupunktowej analizy wartości brzegowych (x = 10, x = 11) mogą nie pozwolić na wykrycie defektu. Jeśli natomiast zostanie zastosowana wartość x = 9 uzyskana w wyniku analizy trójpunktowej, defekt prawdopodobnie zostanie wykryty.

### 4.2.3. Testowanie w oparciu o tablicę decyzyjną

Tablice decyzyjne służą do testowania implementacji wymagań systemowych, które określają, w jaki sposób różne kombinacje warunków powodują uzyskanie różnych wyników. Za pomocą tablic decyzyjnych można sprawnie odzwierciedlać złożone mechanizmy logiczne, takie jak reguły biznesowe.

Przy opracowywaniu tablic decyzyjnych określa się warunki i wynikające z nich akcje systemu, które tworzą wiersze tablicy. Każda kolumna odpowiada regule decyzyjnej, która określa unikatową kombinację warunków wraz z powiązanymi akcjami. W ograniczonych tablicach decyzyjnych wszystkie wartości warunków i akcji (z wyjątkiem warunków nieistotnych lub niemożliwych do spełnienia; patrz poniżej) przedstawia się jako wartości logiczne (prawda/fałsz). Alternatywą dla powyższego wariantu są uogólnione tablice decyzyjne, w których niektóre lub wszystkie warunki i akcje mogą również przyjmować wiele wartości (takich jak przedziały liczb, klasy równoważności czy wartości dyskretne).

Notacja warunków jest następująca: „P" (prawda) oznacza, że warunek został spełniony, „F" (fałsz) oznacza, że warunek nie został spełniony, „—" oznacza, że wartość warunku nie ma znaczenia dla wyniku akcji, a „nd" („nie dotyczy") oznacza, że warunek nie występuje w przypadku danej reguły. W przypadku akcji „X" oznacza, że akcja powinna zostać wykonana, a puste pole — że nie powinna zostać wykonana. Mogą być również stosowane inne notacje.

Pełna tablica decyzyjna zawiera tyle kolumn, ile jest niezbędne do pokrycia wszystkich kombinacji warunków. Tablicę można uprościć poprzez usunięcie kolumn zawierających kombinacje warunków, które są niemożliwe do spełnienia. Ponadto można ją zminimalizować poprzez scalenie kolumn, w których część warunków nie ma wpływu na wynik, w jedną kolumnę. Należy jednak zaznaczyć, że algorytmy minimalizowania tablic decyzyjnych nie są przedmiotem niniejszego sylabusa.

W przypadku testowania w oparciu o tablicę decyzyjną elementami pokrycia są kolumny zawierające możliwe do spełnienia kombinacje warunków. Warunkiem uzyskania stuprocentowego pokrycia przy korzystaniu z tej techniki jest sprawdzenie za pomocą przypadków testowych wszystkich powyższych kolumn. Pokrycie mierzy się jako iloraz liczby sprawdzonych kolumn przez łączną liczbę kolumn zawierających możliwe do spełnienia warunki, a uzyskana wartość jest wyrażona w procentach.

Zaletą testowania w oparciu o tablicę decyzyjną jest systematyczne podejście umożliwiające zidentyfikowanie wszystkich kombinacji warunków, które w innym przypadku mogłyby zostać przeoczone. Ponadto metoda ta pomaga znaleźć ewentualne luki lub sprzeczności w wymaganiach. Jeśli występuje duża liczba warunków, sprawdzenie wszystkich reguł decyzyjnych może być czasochłonne, ponieważ liczba reguł rośnie wykładniczo wraz z liczbą warunków. W takim przypadku w celu zmniejszenia liczby reguł wymagających sprawdzenia można zminimalizować tablicę decyzyjną lub zastosować podejście oparte na ryzyku.

### 4.2.4. Testowanie przejść pomiędzy stanami

Diagram przejść pomiędzy stanami umożliwia modelowanie zachowania systemu poprzez zobrazowanie jego możliwych stanów i poprawnych przejść między nimi. Przejście jest inicjowane przez zdarzenie, które może być dodatkowo kwalifikowane przez warunek dozoru. Przyjmuje się, że przejścia są natychmiastowe i mogą niekiedy powodować wykonanie przez oprogramowanie określonej akcji. Typowa notacja stosowana do oznaczania przejść ma postać „zdarzenie [warunek dozoru] / akcja”, przy czym warunki dozoru i akcje można pominąć, jeśli nie istnieją lub są nieistotne z punktu widzenia testera.

Tablica stanów jest modelem równoważnym diagramowi przejść pomiędzy stanami. Jej wiersze odpowiadają stanom, a kolumny — zdarzeniom (wraz z ewentualnymi warunkami dozoru). Wpisy w tabeli (komórki) odpowiadają przejściom i zawierają informacje o stanie docelowym oraz o akcjach wynikających z przejść (jeśli zostały zdefiniowane). W przeciwieństwie od diagramu przejść pomiędzy stanami tablica stanów wyraźnie wskazuje niepoprawne przejścia, którym odpowiadają puste komórki.

Przypadek testowy oparty na diagramie przejść pomiędzy stanami lub tablicy stanów jest zwykle wyrażany w postaci sekwencji zdarzeń, która powoduje wykonanie sekwencji zmian stanu (oraz akcji, jeśli są wymagane). Jeden przypadek testowy może pokrywać (i zwykle pokrywa) kilka przejść pomiędzy stanami.

Istnieje wiele kryteriów pokrycia, które można wykorzystać w testowaniu przejść pomiędzy stanami. W niniejszym sylabusie omówiono trzy z nich.

W przypadku **pokrycia wszystkich stanów** elementami pokrycia są właśnie stany. Warunkiem uzyskania stuprocentowego pokrycia wszystkich stanów jest przejście za pomocą przypadków testowych przez wszystkie istniejące stany. Pokrycie mierzy się jako iloraz liczby odwiedzonych stanów przez łączną liczbę stanów, a uzyskana wartość jest wyrażona w procentach.

W przypadku **pokrycia poprawnych przejść** (zwanego także pokryciem 0-przełączeń) elementami pokrycia są pojedyncze poprawne przejścia. Warunkiem uzyskania stuprocentowego pokrycia poprawnych przejść jest wykonanie za pomocą przypadków testowych wszystkich poprawnych przejść. Pokrycie mierzy

się jako iloraz liczby wykonanych poprawnych przejść przez łączną liczbę poprawnych przejść, a uzyskana wartość jest wyrażona w procentach.

W przypadku **pokrycia wszystkich przejść** elementami pokrycia są wszystkie przejścia wskazane w tablicy stanów. Warunkiem uzyskania stuprocentowego pokrycia wszystkich przejść jest wykonanie za pomocą przypadków testowych wszystkich poprawnych przejść oraz podjęcie próby wykonania niepoprawnych przejść. Testowanie tylko jednego niepoprawnego przejścia w jednym przypadku testowym pozwala uniknąć maskowania defektów, czyli sytuacji, w której jeden defekt uniemożliwia wykrycie innego defektu. Pokrycie mierzy się jako iloraz liczby poprawnych i niepoprawnych przejść, które zostały wykonane lub w przypadku których podjęto próbę wykonania za pomocą przypadków testowych przez łączną liczbę poprawnych i niepoprawnych przejść, a uzyskana wartość jest wyrażona w procentach.

Pokrycie wszystkich stanów jest słabszym kryterium niż pokrycie poprawnych przejść, ponieważ można je zwykle osiągnąć bez wykonania wszystkich przejść. Pokrycie poprawnych przejść jest najczęściej stosowanym kryterium pokrycia. Uzyskanie pełnego pokrycia poprawnych przejść gwarantuje pełne pokrycie wszystkich stanów, a uzyskanie pełnego pokrycia wszystkich przejść gwarantuje zarówno pełne pokrycie wszystkich stanów, jak i pełne pokrycie poprawnych przejść, w związku z czym powinno być wymogiem minimalnym w przypadku oprogramowania o newralgicznym znaczeniu dla działalności przedsiębiorstwa, a także oprogramowania krytycznego ze względów bezpieczeństwa.

## 4.3. Białoskrzynkowe techniki testowania

Z uwagi na ich popularność i prostotę w niniejszym podrozdziale skupiono się na dwóch białoskrzynkowych technikach testowania związanych z kodem. Są to:

- testowanie instrukcji;
- testowanie gałęzi.

Istnieją również bardziej rygorystyczne techniki, których używa się w systemach krytycznych ze względów bezpieczeństwa, w systemach o newralgicznym znaczeniu dla działalności przedsiębiorstwa oraz w środowiskach wymagających wysokiego poziomu integralności w celu uzyskania pełniejszego pokrycia kodu. Istnieją także białoskrzynkowe techniki testowania używane na wyższych poziomach testów (np. w ramach testowania API) lub korzystające z pokrycia niezwiązanego z kodem (np. pokrycia neuronów w testowaniu sieci neuronowych). Nie są one jednak przedmiotem niniejszego sylabusa.

### 4.3.1. Testowanie instrukcji i pokrycie instrukcji kodu

W przypadku testowania instrukcji elementami pokrycia są instrukcje wykonywalne. Technika ta służy do projektowania przypadków testowych, które sprawdzają instrukcje w kodzie do momentu osiągnięcia akceptowalnego poziomu pokrycia. Pokrycie mierzy się jako iloraz liczby instrukcji sprawdzonych za pomocą przypadków testowych przez łączną liczbę instrukcji wykonywalnych w kodzie, a uzyskana wartość jest wyrażona w procentach.

Uzyskanie stuprocentowego pokrycia instrukcji kodu gwarantuje, że każda instrukcja wykonywalna została sprawdzona co najmniej raz. Oznacza to w szczególności wykonanie każdej instrukcji zawierającej defekt, co może spowodować awarię potwierdzającą istnienie defektu. Należy jednak zaznaczyć, że sprawdzenie instrukcji za pomocą przypadku testowego nie zawsze powoduje wykrycie defektu — istnieje na przykład ryzyko niewykrycia defektów, które są zależne od danych (takich jak dzielenie przez zero, które powoduje awarię tylko w przypadku nadania mianownikowi wartości zerowej). Ponadto uzyskanie stuprocentowego

pokrycia instrukcji kodu nie gwarantuje, że została przetestowana cała logika decyzyjna, ponieważ testy mogą na przykład nie sprawdzać wszystkich gałęzi kodu (patrz sekcja 4.3.2).

### 4.3.2. Testowanie gałęzi i pokrycie gałęzi

Gałąź reprezentuje przepływ sterowania między dwoma wierzchołkami w diagramie przepływu sterowania przedstawiającym możliwe sekwencje wykonywania instrukcji kodu źródłowego w przedmiocie testów. Każdy przepływ sterowania może odbywać się albo bezwarunkowo (kod liniowy), albo warunkowo (wynik decyzji).

W przypadku testowania gałęzi elementami pokrycia są gałęzie, a celem jest zaprojektowanie przypadków testowych sprawdzających gałęzie kodu do momentu osiągnięcia akceptowalnego poziomu pokrycia. Pokrycie mierzy się jako iloraz liczby gałęzi sprawdzonych za pomocą co najmniej jednego przypadku testowego przez łączną liczbę gałęzi, a uzyskana wartość jest wyrażona w procentach.

Uzyskanie stuprocentowego pokrycia gałęzi oznacza, że wszystkie gałęzie kodu — zarówno bezwarunkowe, jak i warunkowe — zostały sprawdzone za pomocą przypadków testowych. Gałęzie warunkowe zwykle odpowiadają wynikowi „prawda” lub „fałsz” decyzji „IF … THEN”, wynikowi instrukcji SWITCH/CASE bądź decyzji o wyjściu z pętli lub dalszym wykonywaniu pętli. Należy jednak zaznaczyć, że sprawdzenie gałęzi za pomocą przypadku testowego nie zawsze powoduje wykrycie defektu — istnieje na przykład ryzyko niewykrycia defektów, które wymagają wykonania konkretnej ścieżki w kodzie.

Pokrycie gałęzi subsumuje pokrycie instrukcji kodu, co oznacza, że każdy zbiór przypadków testowych osiągający stuprocentowe pokrycie gałęzi osiąga również stuprocentowe pokrycie instrukcji kodu (ale nie odwrotnie).

### 4.3.3. Korzyści wynikające z testowania białoskrzynkowego

Zasadniczą zaletą wszystkich technik białoskrzynkowych jest fakt, że podczas testowania uwzględniana jest cała implementacja oprogramowania, co ułatwia wykrywanie defektów nawet w przypadku, gdy specyfikacja oprogramowania jest niejednoznaczna, nieaktualna lub niekompletna. Wadą jest z kolei fakt, że w przypadku niezaimplementowania w oprogramowaniu jednego lub kilku wymagań testowanie białoskrzynkowe może nie wykryć spowodowanych tym defektów w postaci pominięć (Watson 1996).

Techniki białoskrzynkowe mogą być stosowane w testowaniu statycznym (np. podczas próbnych przebiegów kodu - ang. *dry runs*). Sprawdzają się również w przeglądach kodu, który nie jest jeszcze gotowy do uruchomienia (Hetzel 1988), a także pseudokodu oraz mechanizmów logicznych, które można odwzorować w postaci diagramu przepływu sterowania.

Wykonanie jedynie testowania czarnoskrzynkowego nie pozwala zmierzyć faktycznego pokrycia kodu, natomiast miary pokrycia stosowane w technikach białoskrzynkowych zapewniają obiektywny pomiar pokrycia i dostarczają niezbędnych informacji umożliwiających wygenerowanie dodatkowych testów w celu jego zwiększenia, co ostatecznie przekłada się na wzrost zaufania do kodu.

## 4.4. Techniki testowania oparte na doświadczeniu

W kolejnych sekcjach omówiono następujące powszechnie stosowane techniki testowania oparte na doświadczeniu:

- zgadywanie błędów;
- testowanie eksploracyjne;
- testowanie w oparciu o listę kontrolną.

### 4.4.1. Zgadywanie błędów

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

- dotychczasowego działania aplikacji;
- typowych błędów popełnianych przez programistów i typów wynikających z nich defektów;
- rodzajów awarii, które wystąpiły w innych podobnych aplikacjach.

Zasadniczo błędy, defekty i awarie mogą być związane z: danymi wejściowymi (np. nieprzyjęcie poprawnych danych wejściowych bądź błędne lub brakujące parametry), danymi wyjściowymi (np. nieprawidłowy format lub nieprawidłowy rezultat), logiką (np. brakujące przypadki lub nieprawidłowy operator), obliczeniami (np. niepoprawny argument operacji lub błędne obliczenie), interfejsami (np. niezgodność parametrów lub typów) bądź danymi (np. niepoprawne zainicjowanie lub niewłaściwy typ danych).

Przykładem metodycznego podejścia do zgadywania błędów są ataki usterek. Technika ta wymaga od testera stworzenia lub uzyskania listy potencjalnych błędów, defektów i awarii, a następnie zaprojektowania testów pozwalających zidentyfikować defekty związane z błędami z listy, uwidocznić defekty z listy bądź spowodować awarie z listy. Listy takie można opracowywać na podstawie własnego doświadczenia, danych dotyczących defektów i awarii oraz powszechnej wiedzy na temat przyczyn awarii oprogramowania.

Więcej informacji na temat zgadywania błędów i ataków usterek zawierają pozycje (Whittaker 2002, Whittaker 2003, Andrews 2006).

### 4.4.2. Testowanie eksploracyjne

Testowanie eksploracyjne polega na równoczesnym projektowaniu, wykonywaniu i dokonywaniu oceny testów w czasie, gdy tester zapoznaje się z przedmiotem testów. Proces ten dostarcza wiedzy na temat przedmiotu testów, a także pozwala tworzyć ukierunkowane testy umożliwiające jego dokładniejsze zbadanie oraz testy dotyczące obszarów dotychczas nieprzetestowanych.

Testowanie eksploracyjne jest czasami przeprowadzane metodą tzw. testowania w sesjach, która pozwala uporządkować cały proces. W ramach testowania w sesjach testowanie eksploracyjne odbywa się w ściśle określonym przedziale czasu, a tester prowadzi testy zgodnie z kartą opisu testu (zawierającą cele testów). Po zakończeniu sesji testowej zwykle odbywa się spotkanie podsumowujące, którego elementem jest dyskusja między testerem a interesariuszami zainteresowanymi wynikami sesji testowej. W ramach tego podejścia cele testów można potraktować jako warunki testowe wysokiego poziomu. W trakcie sesji

testowej identyfikuje się i sprawdza elementy pokrycia. Do dokumentowania wykonywanych kroków i uzyskiwanych informacji służą arkusze sesji testowych.

Testowanie eksploracyjne jest przydatne w przypadku niepełnych lub niewłaściwie sporządzonych specyfikacji bądź w przypadku testowania pod presją czasu. Ponadto może być uzupełnieniem innych, bardziej formalnych technik testowania. Wśród czynników zwiększających skuteczność testowania eksploracyjnego można wymienić doświadczenie i wiedzę merytoryczną testera oraz wysoki poziom niezbędnych umiejętności, takich jak umiejętność analizy, ciekawość i kreatywność (patrz sekcja 1.5.1).

W ramach testowania eksploracyjnego można korzystać także z innych technik (np. techniki podziału na klasy równoważności). Więcej informacji na temat testowania eksploracyjnego można znaleźć w (Kaner 1999, Whittaker 2009, Hendrickson 2013).

### 4.4.3. Testowanie w oparciu o listę kontrolną

W ramach testowania w oparciu o listę kontrolną testerzy projektują, implementują i wykonują testy tak, aby pokryć warunki testowe wymienione na liście kontrolnej. Listy kontrolne można opracowywać na podstawie własnego doświadczenia, znajomości oczekiwań użytkowników lub wiedzy na temat przyczyn i objawów awarii oprogramowania. Należy jednak pamiętać, że nie powinny one zawierać elementów, które można sprawdzić automatycznie, które lepiej jest wykorzystać jako kryteria wejścia/wyjścia lub które są zbyt ogólne (Brykczynski 1999).

Elementy listy kontrolnej są często formułowane jako pytania, przy czym powinno być możliwe bezpośrednie sprawdzenie każdego z nich z osobna. Elementy te mogą dotyczyć wymagań, właściwości interfejsu graficznego, charakterystyk jakościowych lub innego rodzaju warunków testowych. Listy kontrolne można tworzyć na potrzeby różnych typów testów, w tym na potrzeby testowania funkcjonalnego i niefunkcjonalnego, czego przykładem jest 10 heurystyk w zakresie testowania użyteczności (Nielsen 1994).

Z czasem niektóre pozycje listy kontrolnej mogą stopniowo tracić skuteczność, ponieważ programiści uczą się unikać popełniania tych samych pomyłek. Ponadto może być konieczne dodanie nowych pozycji odzwierciedlających nowo wykryte defekty o dużej krytyczności. Z powyższych powodów listy kontrolne powinny być regularnie aktualizowane na podstawie analizy defektów, jednak należy zachować ostrożność, aby nie stały się one zbyt obszerne (Gawande 2009).

W sytuacji, w której brakuje szczegółowych przypadków testowych, testowanie w oparciu o listę kontrolną zapewnia niezbędne wytyczne i pozwala uzyskać pewien stopień spójności testowania. Jeśli listy kontrolne mają charakter wysokopoziomowy, podczas faktycznego testowania może występować pewna zmienność przekładająca się na większe pokrycie, ale kosztem mniejszej powtarzalności.

## 4.5. Podejścia do testowania oparte na współpracy

Każda z wyżej wymienionych technik (patrz podrozdziały 4.2, 4.3 i 4.4) ma określony cel związany z wykrywaniem defektów, natomiast podejścia oparte na współpracy koncentrują się również na unikaniu defektów poprzez zapewnienie współpracy i wymiany informacji.

### 4.5.1. Wspólne pisanie historyjek użytkownika

Historyjka użytkownika reprezentuje cechę systemu wartościową dla użytkownika bądź nabywcy systemu lub oprogramowania. W historyjkach użytkownika wyróżnia się trzy kluczowe aspekty (Jeffries 2000) nazywane łącznie „3 C" od pierwszych liter ich angielskich nazw. Są to:

- karta (ang. *card*), czyli nośnik zawierający opis historyjki użytkownika (np. karta katalogowa bądź wpis w kartotece elektronicznej);
- rozmowa (ang. *conversation*), która wyjaśnia, w jaki sposób będzie używane oprogramowanie (w postaci udokumentowanej lub słownej);
- potwierdzenie (ang. *confirmation*), czyli kryteria akceptacji (patrz sekcja 4.5.2).

Najczęściej stosowany format historyjki użytkownika ma postać zdania „Jako [rola] chcę [zakładany cel do osiągnięcia], abym mógł/mogła [wartość biznesowa uzyskiwana w kontekście danej roli]", po którym następują kryteria akceptacji.

Podczas wspólnego tworzenia historyjki użytkownika można korzystać z technik takich jak burza mózgów czy tworzenie map myśli. Dzięki współpracy zespół może wypracować wspólną wizję tego, co należy dostarczyć, z uwzględnieniem trzech punktów widzenia: biznesowego, programistycznego i testowego.

Dobre historyjki użytkownika powinny być niezależne, negocjowalne, wartościowe, możliwe do oszacowania, zwięzłe i testowalne (ang. *Independent, Negotiable, Valuable, Estimable, Small, Testable — INVEST*). Jeśli interesariusz nie wie, jak należy przetestować daną historyjkę użytkownika, może to wskazywać, że nie jest ona wystarczająco jednoznaczna lub nie odzwierciedla kwestii istotnych dla tego interesariusza. Nie można też jednak wykluczyć, że interesariusz po prostu potrzebuje pomocy w testowaniu (Wake 2003).

### 4.5.2. Kryteria akceptacji

Kryteria akceptacji związane z historyjką użytkownika to warunki, jakie muszą zostać spełnione, aby implementacja tej historyjki została zaakceptowana przez interesariuszy. Z tej perspektywy kryteria akceptacji można potraktować jako warunki testowe, których spełnienie powinno zostać sprawdzone w trakcie testów. Kryteria akceptacji powstają zwykle w wyniku rozmowy (patrz sekcja 4.5.1).

Celem kryteriów akceptacji jest:

- określenie zakresu historyjki użytkownika;
- osiągnięcie konsensusu wśród interesariuszy;
- opisanie pozytywnych i negatywnych scenariuszy;
- stworzenie podstawy do testowania akceptacyjnego historyjki użytkownika (patrz sekcja 4.5.3);
- umożliwienie dokładnego planowania i szacowania.

Kryteria akceptacji związane z historyjką użytkownika można pisać na kilka sposobów. Dwa najpopularniejsze formaty to:

- format ukierunkowany na scenariusze (np. format Given/When/Then stosowany w wytwarzaniu sterowanym zachowaniem; patrz sekcja 2.1.3);

- format ukierunkowany na reguły (np. lista weryfikacyjna w punktach lub przypisanie danych wejściowych do danych wyjściowych w formie tabelarycznej).

Większość kryteriów akceptacji można udokumentować w jednym z dwóch powyższych formatów, ale zespół może również skorzystać z innego, niestandardowego formatu pod warunkiem, że kryteria będą należycie zdefiniowane i jednoznaczne.

### 4.5.3. Wytwarzanie sterowane testami akceptacyjnymi (ATDD)

Wytwarzanie sterowane testami akceptacyjnymi to podejście typu „najpierw test" (patrz sekcja 2.1.3), zgodnie z którym przypadki testowe tworzy się przed zaimplementowaniem historyjki użytkownika. Tworzeniem przypadków testowych zajmują się członkowie zespołu mający różne punkty widzenia, na przykład klienci, programiści i testerzy (Adzic 2009). Uzyskane w ten sposób przypadki testowe mogą być następnie wykonywane manualnie lub automatycznie.

Pierwszym krokiem jest przeprowadzenie warsztatów tworzenia specyfikacji, podczas których członkowie zespołu analizują, omawiają i piszą historyjkę użytkownika oraz związane z nią kryteria akceptacji (o ile nie zostały jeszcze zdefiniowane). W ramach tego procesu korygowane są wszelkie braki, niejednoznaczności lub defekty występujące w historyjce użytkownika. Kolejnym krokiem jest stworzenie przypadków testowych, przy czym czynność ta może być wykonywana przez cały zespół lub indywidualnie przez testera. Przypadki testowe są tworzone w oparciu o kryteria akceptacji i mogą być traktowane jako przykłady sposobu działania oprogramowania. Dzięki temu zespół może poprawnie zaimplementować historyjkę użytkownika.

Z uwagi na to, że przykłady i testy są takie same, pojęcia te bywają często używane zamiennie. Podczas projektowania testów można stosować techniki testowania opisane w podrozdziałach 4.2, 4.3 i 4.4.

Pierwsze przypadki testowe są z reguły przypadkami pozytywnymi, co oznacza, że mają na celu potwierdzenie prawidłowego zachowania — bez wyjątków lub warunków błędów — i odzwierciedlają sekwencję czynności wykonywaną w sytuacji, w której wszystko działa zgodnie z oczekiwaniami. Po wykonaniu pozytywnych przypadków testowych zespół powinien przystąpić do testowania negatywnego, a na koniec powinien również zadbać o pokrycie niefunkcjonalnych charakterystyk jakościowych (takich jak wydajność czy użyteczność). Przypadki testowe powinny być wyrażone w sposób zrozumiały dla interesariuszy. Zazwyczaj zawierają one zdania w języku naturalnym określające niezbędne warunki wstępne (jeśli mają zastosowanie), dane wejściowe oraz warunki wyjściowe.

Przypadki testowe muszą pokrywać wszystkie charakterystyki historyjki użytkownika, ale nie powinny wykraczać poza jej zakres, natomiast kryteria akceptacji mogą opisywać szczegółowo niektóre kwestie opisane w historyjce. Ponadto należy unikać sytuacji, w których dwa przypadki testowe opisują tę samą charakterystykę historyjki użytkownika.

Jeśli przypadki testowe zostaną zarejestrowane w formacie obsługiwanym przez strukturę do testów automatycznych, programiści mogą zautomatyzować ich wykonywanie poprzez opracowanie kodu pomocniczego w trakcie implementowania funkcjonalności opisanej w historyjce użytkownika. Testy akceptacyjne stają się wówczas wykonywalnymi wymaganiami.

# 5. Zarządzanie czynnościami testowymi — 335 minut

**Słowa kluczowe**

analiza ryzyka, identyfikacja ryzyka, kontrola ryzyka, kryteria wejścia, kryteria wyjścia, kwadranty testowe, łagodzenie ryzyka, monitorowanie ryzyka, monitorowanie testów, nadzór nad testami, ocena ryzyka, piramida testów, plan testów, planowanie testów, podejście do testowania, poziom ryzyka, raport o defekcie, raport o postępie testów, ryzyko, ryzyko produktowe, ryzyko projektowe, sumaryczny raport z testów, testowanie oparte na ryzyku, zarządzanie defektami, zarządzanie ryzykiem

**Cele nauczania w rozdziale 5:**

### 5.1 Planowanie testów

FL-5.1.1 (K2) Kandydat omawia na przykładach cel i treść planu testów.

FL-5.1.2 (K1) Kandydat rozpoznaje, jaki jest wkład testera w planowanie iteracji i wydań.

FL-5.1.3 (K2) Kandydat porównuje i zestawia ze sobą kryteria wejścia i kryteria wyjścia.

FL-5.1.4 (K3) Kandydat oblicza pracochłonność testowania przy użyciu technik szacowania.

FL-5.1.5 (K3) Kandydat stosuje priorytetyzację przypadków testowych.

FL-5.1.6 (K1) Kandydat pamięta pojęcia związane z piramidą testów.

FL-5.1.7 (K2) Kandydat podsumowuje kwadranty testowe oraz ich relację do poziomów testów i typów testów.

### 5.2 Zarządzanie ryzykiem

FL-5.2.1 (K1) Kandydat określa poziom ryzyka na podstawie prawdopodobieństwa ryzyka i wpływu ryzyka.

FL-5.2.2 (K2) Kandydat rozróżnia ryzyka projektowe i produktowe.

FL-5.2.3 (K2) Kandydat wyjaśnia potencjalny wpływ analizy ryzyka produktowego na staranność i zakres testowania.

FL-5.2.4 (K2) Kandydat wyjaśnia, jakie środki można podjąć w odpowiedzi na przeanalizowane ryzyka produktowe.

### 5.3 Monitorowanie testów, nadzór nad testami i ukończenie testów

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

FL-5.3.2 (K2) Kandydat podsumowuje cele i treść raportów z testów oraz wskazuje ich odbiorców.

FL-5.3.3 (K2) Kandydat omawia na przykładach sposób przekazywania informacji o statusie testowania.

### 5.4 Zarządzanie konfiguracją

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

### 5.5 Zarządzanie defektami

FL-5.5.1 (K3) Kandydat sporządza raport o defekcie.

# 5.1. Planowanie testów

## 5.1.1. Cel i treść planu testów

W planie testów opisuje się cele, zasoby i procesy związane z projektem testowania. Plan testów:

- pozwala udokumentować sposób i harmonogram osiągania celów testów;
- pomaga zagwarantować, że wykonywane czynności testowe spełnią ustalone kryteria;
- służy do wymiany informacji z członkami zespołu i innymi interesariuszami;
- pozwala wykazać, że testowanie będzie odbywać się zgodnie z dotychczasową polityką testów i strategią testów (bądź uzasadnić ewentualne odstępstwa od nich).

Planowanie testów pozwala odpowiednio ukierunkować sposób myślenia testerów i zmusza ich do stawienia czoła przyszłym wyzwaniom związanym z ryzykami, harmonogramami, relacjami interpersonalnymi, narzędziami, kosztami, nakładami pracy itd. Proces przygotowywania planu testów jest dobrą okazją do przemyślenia nakładów pracy, jakie będą niezbędne do osiągnięcia celów projektu testowania.

Typowe zagadnienia poruszane w planie testów to między innymi:

- kontekst testowania (np. zakres, cele testów, ograniczenia i podstawa testów);
- założenia i ograniczenia projektu testowania;
- interesariusze (np. role, obowiązki, istotność z punktu widzenia testowania, potrzeby w zakresie rekrutacji i szkoleń);
- wymiana informacji (np. formy i częstotliwość wymiany informacji, szablony dokumentów);
- rejestr ryzyk (np. ryzyk produktowych i projektowych);
- podejście do testowania (np. poziomy testów, typy testów, techniki testowania, produkty pracy związane z testami, kryteria wejścia i wyjścia, niezależność testowania, gromadzone metryki, wymagania dotyczące danych testowych, wymagania dotyczące środowiska testowego, odstępstwa od polityki testów i strategii testów obowiązujących w organizacji);
- budżet i harmonogram.

Dokładniejsze informacje na temat planu testów i jego treści zawiera standard ISO/IEC/IEEE 29119-3.

## 5.1.2. Wkład testera w planowanie iteracji i wydań

W iteracyjnych cyklach wytwarzania oprogramowania występują zwykle dwa rodzaje planowania: planowanie wydań i planowanie iteracji.

Planowanie wydań odbywa się w perspektywie wprowadzenia produktu do eksploatacji i obejmuje definiowanie/redefiniowanie backlogu produktu, a w pewnych przypadkach również doprecyzowanie większych historyjek użytkownika poprzez podzielenie ich na szereg mniejszych historyjek. Opracowany w ten sposób plan jest też podstawą podejścia do testowania i planu testów we wszystkich iteracjach. Testerzy zaangażowani w planowanie wydań piszą testowalne historyjki użytkownika i kryteria akceptacji (patrz podrozdział 4.5), uczestniczą w analizach ryzyka projektowego i produktowego (patrz

podrozdział 5.2), szacują pracochłonność testowania związanego z historyjkami użytkownika (patrz sekcja 5.1.4), ustalają podejście do testowania oraz planują testowanie w związku z danym wydaniem.

Planowanie iteracji odbywa się z kolei w perspektywie zakończenia pojedynczej iteracji produktu i dotyczy backlogu tej iteracji. Testerzy zaangażowani w planowanie iteracji uczestniczą w szczegółowej analizie ryzyka związanego z historyjkami użytkownika, określają testowalność historyjek użytkownika, dzielą historyjki użytkownika na zadania (w szczególności zadania testowe), szacują pracochłonność wszystkich zadań testowych oraz identyfikują i doprecyzowują aspekty funkcjonalne i niefunkcjonalne przedmiotu testów.

### 5.1.3. Kryteria wejścia i kryteria wyjścia

Kryteria wejścia określają warunki wstępne, które muszą zostać spełnione przed rozpoczęciem danej czynności. W przypadku niespełnienia kryteriów wejścia wykonanie tej czynności może być trudniejsze oraz bardziej czasochłonne, kosztowne i ryzykowne. Kryteria wyjścia określają warunki, które muszą zostać spełnione, aby można było uznać wykonywanie danej czynności za zakończone. Kryteria wejścia i kryteria wyjścia należy zdefiniować w odniesieniu do każdego poziomu testów (kryteria te różnią się w zależności od celów testów).

Typowe kryteria wejścia to między innymi: dostępność zasobów (takich jak pracownicy, narzędzia, środowiska, dane testowe, budżet i czas), dostępność testaliów (takich jak podstawa testów, testowalne wymagania, historyjki użytkownika i przypadki testowe) oraz początkowy poziom jakości przedmiotu testów (np. pomyślne przejście wszystkich testów dymnych).

Typowe kryteria wyjścia to między innymi: miary staranności (np. osiągnięty poziom pokrycia, liczba nieusuniętych defektów, gęstość defektów lub liczba niezaliczonych przypadków testowych) oraz kryteria ukończenia (np. wykonanie zaplanowanych testów, wykonanie testowania statycznego, zgłoszenie wszystkich wykrytych defektów bądź zautomatyzowanie wszystkich testów regresji).

Za poprawne kryterium wyjścia można także uznać przekroczenie terminu lub budżetu. Zakończenie testowania w takich okolicznościach — nawet w przypadku niespełnienia innych kryteriów wyjścia — może być dopuszczalne, o ile interesariusze znają i akceptują ryzyko związane z wydaniem systemu bez dalszego testowania.

W modelu zwinnego wytwarzania oprogramowania kryteria wyjścia są zwykle nazywane definicją ukończenia (ang. *Definition of Done*) i określają obiektywne metryki, zgodnie z którymi zespół może uznać dany element za nadający się do przekazania do eksploatacji. Kryteria wejścia, które musi spełnić historyjka użytkownika, aby można było rozpocząć prace programistyczne i/lub czynności związane z testowaniem, są nazywane definicją gotowości (ang. *Definition of Ready*).

### 5.1.4. Techniki szacowania

Szacowanie pracochłonności testów polega na przewidywaniu nakładów pracy związanych z testowaniem, które są niezbędne do osiągnięcia celów projektu testowania. Ważne jest przy tym uświadomienie interesariuszom, że oszacowania dokonuje się na podstawie szeregu założeń, w związku z czym jest ono zawsze obarczone błędem szacowania. Oszacowania są zwykle dokładniejsze w przypadku mniejszych zadań, dlatego przy szacowaniu dużych zadań należy rozbić takie zadania na szereg mniejszych elementów, co ułatwia określenie przewidywanego nakładu pracy.

W niniejszym sylabusie opisano następujące cztery techniki szacowania:

**Szacowanie na podstawie proporcji.** W ramach tej techniki opartej na metrykach gromadzi się dane liczbowe z wcześniejszych projektów realizowanych w danej organizacji, co umożliwia ustalenie „standardowych" proporcji współczynników w odniesieniu do podobnych projektów. Współczynniki opisujące własne projekty danej organizacji (określone np. na podstawie danych historycznych) są co do zasady najlepszym źródłem informacji na potrzeby procesu szacowania. W oparciu o powyższe współczynniki standardowe można następnie oszacować pracochłonność nowego projektu. Na przykład jeśli w poprzednim projekcie proporcja pracochłonności wytwarzania do testowania oprogramowania wyniosła 3:2, a w bieżącym projekcie nakłady pracy na wytwarzanie mają wynieść 600 osobodni, to pracochłonność testowania można oszacować na poziomie 400 osobodni.

**Ekstrapolacja.** Podstawą tej techniki opartej na metrykach jest dokonywanie pomiarów na jak najwcześniejszym etapie bieżącego projektu w celu zebrania niezbędnych danych. Po dokonaniu wystarczającej liczby obserwacji pracochłonność pozostałych zadań można określić w przybliżeniu poprzez ekstrapolację dostępnych danych (zwykle z zastosowaniem modelu matematycznego). Metoda ta doskonale sprawdza się w iteracyjnych cyklach wytwarzania oprogramowania, w których zespół może na przykład ekstrapolować pracochłonność testów w nadchodzącej iteracji poprzez wyciągnięcie średniej z ostatnich trzech iteracji.

**Szerokopasmowa technika delficka.** W przypadku tej iteracyjnej techniki eksperckiej eksperci dokonują oszacowań na podstawie posiadanego doświadczenia. Każdy ekspert szacuje nakłady pracy samodzielnie (bez konsultacji z innymi). Następnie wyniki są zbierane i analizowane pod kątem występowania odchyleń wykraczających poza uzgodnione granice, a w przypadku wystąpienia takich odchyleń eksperci wspólnie omawiają bieżące oszacowania. Następnie każdy z ekspertów jest proszony o przygotowanie (również bez konsultacji z innymi ekspertami) nowego oszacowania na podstawie otrzymanych informacji zwrotnych. Proces ten jest powtarzany do momentu osiągnięcia konsensusu. Wariantem szerokopasmowej techniki delfickiej jest tzw. poker planistyczny — metoda powszechnie stosowana w zwinnym wytwarzaniu oprogramowania. W ramach tej metody oszacowań dokonuje się zwykle przy użyciu kart z liczbami odpowiadającymi wielkości nakładów pracy.

**Szacowanie trójpunktowe.** W przypadku tej techniki eksperckiej eksperci przygotowują trzy oszacowania: najbardziej optymistyczne (a), najbardziej prawdopodobne (m) i najbardziej pesymistyczne (b), a ostateczna szacowana wartość (E) jest ich średnią ważoną arytmetyczną. W najpopularniejszym wariancie tej techniki przewidywany nakład pracy oblicza się według wzoru: $E = (a + 4 \times m + b) / 6$. Zaletą tej techniki jest fakt, że umożliwia ona ekspertom obliczenie błędu pomiaru: $SD = (b – a) / 6$. Na przykład jeśli szacowane wartości (w osobogodzinach) wynoszą $a = 6$, $m = 9$ i $b = 18$, to ostateczna szacowana wartość wynosi 10±2 osobogodzin (tj. od 8 do 12 osobogodzin), ponieważ $E = (6 + 4 \times 9 + 18) / 6 = 10$, a $SD = (18 – 6) / 6 = 2$.

Więcej informacji na temat tych i wielu innych technik szacowania testów można znaleźć w (Kan 2003, Koomen 2006, Westfall 2009).

### 5.1.5. Ustalanie priorytetów przypadków testowych

Po opracowaniu specyfikacji przypadków i procedur testowych oraz zgrupowaniu ich w zestawy testowe można utworzyć na ich podstawie harmonogram wykonywania testów, który określa kolejność, w jakiej mają być uruchamiane. Przy ustalaniu priorytetów przypadków testowych należy uwzględnić różne czynniki. Poniżej opisano najczęściej stosowane strategie ustalania priorytetów przypadków testowych.

- Ustalanie priorytetów na podstawie ryzyka. W przypadku tej strategii o kolejności wykonywania testów decydują wyniki analizy ryzyka (patrz sekcja 5.2.3). Jako pierwsze wykonywane są przypadki testowe pokrywające najważniejsze ryzyka.
- Ustalanie priorytetów na podstawie pokrycia. W przypadku tej strategii kolejność wykonywania testów wynika z pokrycia (np. pokrycia instrukcji kodu). Jako pierwsze wykonywane są przypadki testowe pozwalające uzyskać największe pokrycie. W innym wariancie, zwanym ustalaniem priorytetów na podstawie dodatkowego pokrycia, jako pierwszy wykonywany jest przypadek testowy zapewniający największe pokrycie, a w następnej kolejności — przypadki zapewniające największe dodatkowe pokrycie.
- Ustalanie priorytetów na podstawie wymagań. Zgodnie z tą strategią kolejność wykonywania testów ustala się na podstawie priorytetów wymagań, które można powiązać z odpowiadającymi im przypadkami testowymi. Priorytety wymagań określają w tym przypadku interesariusze. Jako pierwsze wykonywane są przypadki testowe związane z najważniejszymi wymaganiami.

O ile jest to możliwe, przypadki testowe powinny być wykonywane w kolejności odpowiadającej poziomom priorytetów — na przykład z wykorzystaniem jednej z wyżej wymienionych strategii ustalania priorytetów. Praktyka ta może jednak nie mieć zastosowania, jeśli przypadki testowe lub testowane przez nie cechy są uzależnione od innych elementów. Przykładem może być sytuacja, gdy przypadek testowy o wyższym priorytecie jest uzależniony od przypadku o niższym priorytecie — wówczas należy w pierwszej kolejności wykonać przypadek o niższym priorytecie.

Przy ustalaniu kolejności wykonywania testów należy również wziąć pod uwagę dostępność zasobów. Wynika to z faktu, że niektóre zasoby — takie jak niezbędne narzędzia lub środowiska testowe bądź pracownicy — mogą być dostępne tylko przez określony czas.

### 5.1.6. Piramida testów

Piramida testów to model odzwierciedlający fakt, że różne testy mogą mieć różną szczegółowość. Model piramidy testów pomaga zespołowi w automatyzowaniu testów i określaniu ich pracochłonności poprzez wskazanie, że różne cele można osiągnąć przy zastosowaniu różnych poziomów automatyzacji. Poszczególne warstwy piramidy odpowiadają grupom testów — im wyższa warstwa, tym mniejsza szczegółowość testów, niższy poziom ich izolacji i dłuższy czas ich wykonywania. Testy umieszczone w najniższej warstwie są małe, odizolowane i szybkie, a do tego sprawdzają niewielką część funkcjonalności, w związku z czym do uzyskania należytego pokrycia potrzebna jest zwykle ich bardzo duża liczba. W najwyższej warstwie znajdują się z kolei złożone i kompleksowe testy wysokiego poziomu. Ich wykonywanie zajmuje więcej czasu niż wykonywanie testów z niższych warstw, a sprawdzany przez nie obszar funkcjonalności jest szeroki, co oznacza, że do uzyskania należytego pokrycia wystarczy tylko kilka takich testów. Liczba i nazewnictwo warstw bywają różne. Na przykład w pierwotnym modelu piramidy testów (Cohn 2009) zdefiniowano trzy warstwy: „testy modułowe”, „testy usług” i „testy interfejsu użytkownika”, a w innym popularnym modelu określono testy modułowe, testy integracyjne (testy integracji modułów) i testy kompleksowe (ang. *end-to-end*). Mogą być również stosowane inne poziomy testów (patrz sekcja 2.2.1).

### 5.1.7. Kwadranty testowe

Kwadranty testowe, które zdefiniował jako pierwszy Brian Marick (Marick 2003, Crispin 2008), służą do grupowania poziomów testów wraz z odpowiednimi typami testów, czynnościami, technikami testowania i produktami pracy w kontekście zwinnego wytwarzania oprogramowania. Model ten ułatwia zarządzanie

testami poprzez wizualizowanie powyższych elementów — co zapewnia uwzględnienie wszystkich odpowiednich typów i poziomów testów w cyklu wytwarzania oprogramowania — oraz uświadamianie testerom, że niektóre typy testów są bardziej istotne niż inne na określonych poziomach testów. Dodatkową zaletą jest możliwość rozróżniania i opisywania poszczególnych typów testów na potrzeby wszystkich interesariuszy, w tym programistów, testerów i przedstawicieli jednostek biznesowych.

W opisywanym modelu testy mogą mieć cel biznesowy lub technologiczny oraz mogą służyć wspieraniu zespołu (np. poprzez ukierunkowanie procesu wytwarzania oprogramowania) lub krytyce produktu (np. poprzez mierzenie jego zachowania względem oczekiwań). Wypadkowa tych dwóch aspektów decyduje o przynależności do jednego z czterech kwadrantów:

- Kwadrant Q1 (cel technologiczny, wspieranie zespołu) obejmuje testy modułowe i testy integracji modułów. Testy te powinny być wykonywane automatycznie i objęte procesem ciągłej integracji.
- Kwadrant Q2 (cel biznesowy, wspieranie zespołu) obejmuje testy funkcjonalne, przykłady, testy oparte na historyjkach użytkownika, prototypy doświadczenia użytkownika, testowanie API oraz symulacje. Testy te sprawdzają spełnienie kryteriów akceptacji i mogą być wykonywane manualnie lub automatycznie.
- Kwadrant Q3 (cel biznesowy, krytyka produktu) obejmuje testowanie eksploracyjne, testowanie użyteczności i testowanie akceptacyjne przez użytkownika. Testy te są ukierunkowane na użytkownika i często wykonywane manualnie.
- Kwadrant Q4 (cel technologiczny, krytyka produktu) obejmuje testy dymne i testy niefunkcjonalne (z wyłączeniem testów użyteczności). Testy te są często wykonywane automatycznie.

## 5.2. Zarządzanie ryzykiem

Organizacje mają do czynienia z wieloma czynnikami wewnętrznymi i zewnętrznymi, które wywołują niepewność co do tego, czy i kiedy zostaną osiągnięte zakładane cele (ISO 31000). Zarządzanie ryzykiem to proces, który umożliwia organizacjom zwiększanie prawdopodobieństwa osiągnięcia celów i podnoszenie jakości produktów, a także budowanie zaufania interesariuszy.

Najważniejsze działania związane z zarządzaniem ryzykiem to:

- analiza ryzyka (obejmująca identyfikację ryzyka i ocenę ryzyka; patrz sekcja 5.2.3);
- kontrola ryzyka (obejmująca łagodzenie ryzyka i monitorowanie ryzyka; patrz sekcja 5.2.4).

Podejście do testowania, zgodnie z którym podstawą doboru czynności testowych i ustalania ich priorytetów oraz zarządzania nimi są analiza ryzyka i kontrola ryzyka, jest nazywane testowaniem opartym na ryzyku.

### 5.2.1. Definicja i atrybuty ryzyka

Ryzyko to potencjalne zdarzenie, niebezpieczeństwo, zagrożenie bądź sytuacja, którego lub której wystąpienie powoduje niekorzystny skutek. Do charakteryzowania ryzyka służą dwa parametry:

- prawdopodobieństwo ryzyka, czyli prawdopodobieństwo wystąpienia danego ryzyka (większe niż zero i mniejsze niż jeden);
- wpływ ryzyka (szkoda), czyli konsekwencje wystąpienia danego ryzyka.

Powyższe dwa parametry wyrażają poziom ryzyka, który jest miarą wielkości danego ryzyka: im wyższy poziom ryzyka, tym ważniejsze jest podjęcie działań zaradczych.

### 5.2.2. Ryzyka projektowe i produktowe

W testowaniu oprogramowania istotne są zasadniczo dwa typy ryzyk: ryzyka projektowe oraz ryzyka produktowe.

**Ryzyka projektowe** są związane z zarządzaniem projektem i nadzorem nad jego realizacją. Do ryzyk projektowych należą:

- problemy organizacyjne (np. opóźnienia w dostawach produktów pracy, niedokładne oszacowania lub cięcia finansowe);
- problemy kadrowe (np. niewystarczające umiejętności, konflikty, problemy z wymianą informacji lub braki kadrowe);
- problemy techniczne (np. nieplanowane rozszerzanie zakresu projektu lub niedostateczna obsługa narzędzi);
- problemy związane z dostawcami (np. niedostarczenie niezbędnego elementu przez zewnętrznego dostawcę lub ogłoszenie upadłości przez firmę świadczącą usługi wsparcia technicznego).

Wystąpienie ryzyka projektowego może mieć wpływ na harmonogram realizacji, budżet lub zakres projektu, a tym samym na możliwość osiągnięcia jego celów.

**Ryzyka produktowe** są związane z charakterystykami jakościowymi produktu (opisanymi np. w modelu jakości określonym przez standard ISO 25010). Przykładami ryzyka produktowego mogą być następujące problemy: brakujące lub niewłaściwe elementy funkcjonalności, niepoprawne obliczenia, awarie podczas wykonywania programu, niedopracowana architektura, nieefektywne algorytmy, zbyt długi czas odpowiedzi, niski poziom doświadczenia użytkownika (ang. *User Experience — UX*) lub podatności na zagrożenia zabezpieczeń. Wystąpienie ryzyka produktowego może pociągać za sobą cały szereg negatywnych konsekwencji, takich jak:

- niezadowolenie użytkowników;
- utrata przychodów, zaufania lub reputacji;
- szkody wyrządzone stronom trzecim;
- wysokie koszty pielęgnacji i przeciążenie struktur wsparcia technicznego;
- odpowiedzialność karna;
- w skrajnych przypadkach — szkody materialne, uszczerbek na zdrowiu, a nawet śmierć.

### 5.2.3. Analiza ryzyka produktowego

Z perspektywy testowania celem analizy ryzyka produktowego jest uzyskanie wiedzy na temat tego rodzaju ryzyka i wykorzystanie jej do ukierunkowania działań wykonywanych podczas testowania w sposób

pozwalający zminimalizować poziom ryzyka rezydualnego (resztkowego). Analizę ryzyka produktowego należy w miarę możliwości rozpocząć na wczesnym etapie cyklu wytwarzania oprogramowania.

Analiza ryzyka produktowego obejmuje identyfikację ryzyka i ocenę ryzyka. Identyfikacja ryzyka polega na wygenerowaniu wyczerpującej listy ryzyk, przy czym interesariusze mogą identyfikować takie ryzyka za pomocą różnych technik i narzędzi, takich jak burza mózgów, warsztaty, wywiady i diagramy przyczynowo-skutkowe. Ocena ryzyka obejmuje z kolei sklasyfikowanie zidentyfikowanych ryzyk, ustalenie prawdopodobieństwa, wpływu i poziomu ryzyka oraz ustalenie priorytetów ryzyk i zaproponowanie sposobów postępowania z nimi. Klasyfikacja ułatwia przypisanie odpowiednich działań związanych z łagodzeniem ryzyka, ponieważ ryzyka zaliczane do tej samej kategorii można zwykle łagodzić podobnymi sposobami.

Oceny ryzyka można dokonać metodą ilościową, jakościową bądź mieszaną. W przypadku podejścia ilościowego poziom ryzyka oblicza się jako iloczyn prawdopodobieństwa ryzyka i wpływu ryzyka, natomiast podejście jakościowe przewiduje ustalanie poziomu ryzyka przy użyciu macierzy ryzyka.

Analiza ryzyka produktowego może mieć wpływ na staranność i zakres testowania. Jej wyniki wykorzystuje się do:

- określenia zakresu wykonywanych testów;
- ustalenia konkretnych poziomów testów i zaproponowania typów testów, jakie mają zostać wykonane;
- wskazania odpowiednich technik testowania i zakładanego poziomu pokrycia;
- oszacowania pracochłonności testowania w odniesieniu do każdego zadania;
- ustalenia priorytetów testowania w sposób sprzyjający jak najwcześniejszemu wykryciu defektów krytycznych;
- ustalenia, czy w celu zmniejszenia ryzyka należy wykonać dodatkowe czynności (poza samym testowaniem).

### 5.2.4. Kontrola ryzyka produktowego

Kontrola ryzyka produktowego obejmuje wszystkie środki podejmowane w odpowiedzi na zidentyfikowane i ocenione ryzyka produktowe. W ramach kontroli ryzyka produktowego wyróżnia się łagodzenie ryzyka i monitorowanie ryzyka. Łagodzenie ryzyka polega na wykonywaniu działań zaproponowanych na etapie oceny ryzyka w celu obniżenia jego poziomu, natomiast celem monitorowania ryzyka jest zapewnienie skuteczności działań związanych z łagodzeniem ryzyka, uzyskanie dalszych informacji pozwalających usprawnić proces oceny ryzyka oraz zidentyfikowanie nowych ryzyk.

Po przeanalizowaniu danego ryzyka proces kontroli ryzyka produktowego przewiduje kilka wariantów, takich jak: łagodzenie ryzyka poprzez testowanie, akceptacja ryzyka, przeniesienie ryzyka oraz plany awaryjne (Veenendaal 2012). Działania, które można podjąć w celu łagodzenia ryzyka produktowego poprzez testowanie, obejmują:

- wytypowanie testerów dysponujących właściwym poziomem doświadczenia i umiejętności — adekwatnie do danego typu ryzyka;
- zapewnienie odpowiedniego poziomu niezależności testowania;
- przeprowadzenie przeglądów i analizy statycznej;

- zastosowanie odpowiednich technik testowania i poziomów pokrycia;
- zastosowanie odpowiednich typów testów uwzględniających charakterystyki jakościowe, których dotyczy ryzyko;
- wykonanie testowania dynamicznego, w tym testowania regresji.

## 5.3. Monitorowanie testów, nadzór nad testami i ukończenie testów

Celem monitorowania testów jest gromadzenie informacji na temat przebiegu procesu testowania. Informacje te są wykorzystywane do oceny postępu testów oraz do pomiaru spełnienia kryteriów wyjścia lub wykonania związanych z nimi zadań testowych (takich jak osiągnięcie zakładanego pokrycia ryzyk produktowych, wymagań lub kryteriów akceptacji).

W ramach nadzoru nad testami informacje uzyskane w procesie monitorowania testów są wykorzystywane do określenia (w formie dyrektyw nadzoru) niezbędnych działań korygujących, które pozwolą uzyskać maksymalną skuteczność i efektywność testowania. Dyrektywy nadzoru mogą obejmować na przykład:

- ponowne ustalenie priorytetów testów w przypadku wystąpienia problemów wynikających ze zidentyfikowanego ryzyka;
- dokonanie ponownej oceny elementu testowego pod kątem spełnienia kryteriów wejścia lub wyjścia w związku z wprowadzeniem poprawek;
- wprowadzenie zmian w harmonogramie testów w związku z opóźnieniami w realizacji środowiska testowego;
- dodanie nowych zasobów w zależności od bieżących potrzeb.

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

### 5.3.1. Metryki stosowane w testowaniu

Metryki dotyczące testów gromadzi się w celu określenia postępu realizacji harmonogramu i budżetu, bieżącej jakości przedmiotu testów oraz skuteczności czynności testowych z punktu widzenia realizacji celów testowania lub celu iteracji. W ramach monitorowania testów zbiera się wiele metryk przydatnych w kontekście nadzoru nad testami i ukończenia testów.

Powszechnie stosowane są między innymi następujące metryki dotyczące testów:

- metryki dotyczące postępu realizacji projektu (np. ukończenie zadań, użycie zasobów, pracochłonność testowania);
- metryki dotyczące postępu testów (np. postęp implementacji przypadków testowych, postęp przygotowania środowiska testowego, liczba wykonanych/niewykonanych i zaliczonych/niezaliczonych przypadków testowych, czas wykonywania testów);
- metryki dotyczące jakości produktów (np. dostępność, czas odpowiedzi, średni czas do awarii);

- metryki dotyczące defektów (np. liczba i priorytety wykrytych/usuniętych defektów, gęstość defektów, odsetek wykrytych defektów);
- metryki dotyczące ryzyka (np. poziom ryzyka rezydualnego);
- metryki dotyczące pokrycia (np. pokrycie wymagań, pokrycie kodu);
- metryki dotyczące kosztów (np. koszty testowania, koszt jakości na poziomie organizacji).

### 5.3.2. Cel, treść i odbiorcy raportów z testów

Raporty z testów służą do podsumowywania i przekazywania informacji na temat testów w trakcie testowania i po jego zakończeniu. Raporty o postępie testów są elementem ciągłego nadzoru nad testami, w związku z czym zawarte w nich informacje muszą być na tyle obszerne, aby w razie potrzeby umożliwić modyfikację harmonogramu testów, przydziału zasobów lub planu testów — na przykład w związku z odstępstwami od planu lub zmieniającymi się okolicznościami. Sumaryczne raporty z testów stanowią podsumowanie określonego etapu testowania (np. poziomu testów, cyklu testowego lub iteracji) i mogą dostarczać informacji na potrzeby dalszego testowania.

W ramach monitorowania i nadzoru zespół testowy sporządza raporty o postępie testów, aby zapewnić interesariuszom stały dostęp do informacji. Raporty takie są zazwyczaj sporządzane regularnie (np. raz dziennie, raz w tygodniu itp.) i zawierają informacje na temat:

- okresu testowania;
- postępu testów (np. wykonania zadań przed terminem lub po terminie), w tym wszelkich istotnych odchyleń;
- utrudnień w testowaniu i sposobów ich obejścia;
- metryk dotyczących testów (przykłady podano w sekcji 5.3.1);
- nowych i zmienionych ryzyk zaobserwowanych w okresie testowania;
- testów zaplanowanych na następny okres.

Sumaryczny raport z testów jest sporządzany na etapie ukończenia testów — po zakończeniu realizacji projektu bądź po wykonaniu testów danego poziomu lub typu, iteracji oraz, w sytuacji idealnej, po spełnieniu kryteriów wyjścia. W raporcie tym wykorzystywane są dane z raportów o postępie testów oraz inne dane.

Typowy sumaryczny raport z testów zawiera między innymi:

- podsumowanie testów;
- ocenę jakości testowania i produktu z punktu widzenia pierwotnego planu testów (tj. celów testów i kryteriów wyjścia);
- informacje o odstępstwach od planu testów (np. o różnicach w stosunku do pierwotnie zakładanego harmonogramu, czasu trwania i nakładu pracy);
- informacje o utrudnieniach w testowaniu i sposobach ich obejścia;
- metryki dotyczące testów określone na podstawie raportów o postępie testów;

- informacje o niezłagodzonych ryzykach i nieusuniętych defektach;
- zdobyte doświadczenia, które są istotne z punktu widzenia testowania.

Poszczególne grupy odbiorców mają różne wymagania co do zakresu informacji podawanych w raportach, a także różne oczekiwania co do stopnia sformalizowania i częstotliwości przekazywania raportów. Wymiana informacji o postępie testów w obrębie zespołu odbywa się zwykle z dużą częstotliwością i na stopie nieformalnej, natomiast raport na temat testowania dotyczący ukończonego projektu jest sporządzany zgodnie z określonym wzorem i przedstawiany tylko raz.

Szablony i przykłady raportów o postępie testów (nazywanych raportami o statusie testów) oraz sumarycznych raportów z testów zawiera standard ISO/IEC/IEEE 29119-3.

### 5.3.3. Przekazywanie informacji o statusie testowania

Optymalny sposób przekazywania informacji o statusie testów zależy od uwarunkowań związanych z zarządzaniem testami, strategii testów przyjętej w danej organizacji i obowiązujących norm prawnych, a w przypadku samoorganizujących się zespołów (patrz sekcja 1.5.2) — również od samego zespołu. Dostępne są między innymi następujące opcje:

- słowna wymiana informacji z członkami zespołu i innymi interesariuszami;
- tablice wskaźników (ang. *dashboard*), np. tablice wskaźników dotyczące ciągłej integracji lub ciągłego dostarczania, tablice zadań i wykresy spalania;
- kanały komunikacji elektronicznej, np. e-mail czy chat;
- dokumentacja w formie elektronicznej;
- formalne raporty z testów (patrz sekcja 5.3.2).

Zależnie od sytuacji można stosować jedną lub kilka z powyższych opcji. W przypadku rozproszonych zespołów, w których bezpośrednie rozmowy nie zawsze są możliwe z uwagi na odległość czy różnicę czasu, lepszym rozwiązaniem może być bardziej formalna wymiana informacji. Ponadto różni interesariusze są zwykle zainteresowani różnymi informacjami, w związku z czym ważne jest odpowiednie dostosowanie treści przekazywanych komunikatów.

## 5.4. Zarządzanie konfiguracją

W kontekście testowania zarządzanie konfiguracją to uporządkowany proces umożliwiający identyfikowanie, nadzorowanie i śledzenie produktów pracy, takich jak: plany testów, strategie testów, warunki testowe, przypadki testowe, skrypty testowe, wyniki testów, dzienniki testów oraz raporty z testów. W ramach tego procesu powyższe produkty pracy są nazywane elementami konfiguracji.

W przypadku złożonego elementu konfiguracji (np. środowiska testowego) zarządzanie konfiguracją pozwala zarejestrować również jego elementy składowe wraz z informacją o relacjach między takimi elementami i ich wersjach. Element konfiguracji, który został zatwierdzony do testowania, staje się konfiguracją bazową i może być modyfikowany wyłącznie w ramach formalnego procesu nadzoru nad zmianami.

W przypadku utworzenia nowej konfiguracji bazowej zmodyfikowane elementy konfiguracji są dokumentowane, aby umożliwić powrót do poprzedniej konfiguracji bazowej w celu odtworzenia wcześniejszych wyników testów.

Zarządzanie konfiguracją umożliwia sprawny przebieg testowania poprzez zagwarantowanie, że:

- wszystkie elementy konfiguracji, w tym elementy testowe (tj. poszczególne elementy przedmiotu testów), są jednoznacznie identyfikowane, objęte kontrolą wersji i śledzeniem zmian oraz powiązane z innymi elementami konfiguracji w sposób pozwalający utrzymać możliwość śledzenia na wszystkich etapach procesu testowego;
- wszystkie zidentyfikowane dokumenty i elementy oprogramowania są przywoływane w sposób jednoznaczny w dokumentacji testów.

Ciągła integracja, ciągłe dostarczanie i ciągłe wdrażanie oraz związane z nimi procesy testowania są zwykle realizowane w ramach zautomatyzowanego potoku DevOps (patrz sekcja 2.1.4), który standardowo obejmuje również zautomatyzowane zarządzanie konfiguracją.

## 5.5. Zarządzanie defektami

Z uwagi na to, że jednym z głównych celów testów jest wykrywanie defektów, niezbędny jest ustalony proces zarządzania defektami. Chociaż w tym podrozdziale jest mowa o „defektach”, zgłaszane anomalie mogą okazać się zarówno rzeczywistymi defektami, jak i czymś zupełnie innym (np. rezultatami fałszywie pozytywnymi lub żądaniami zmiany). Wszelkie wątpliwości w tym zakresie są rozstrzygane podczas rozpatrywania raportów o defektach. Anomalie mogą być zgłaszane w dowolnej fazie cyklu wytwarzania oprogramowania, a sposób ich zgłaszania zależy od konkretnego cyklu. Nieodzownymi elementami procesu zarządzania defektami są: przepływ pracy umożliwiający obsługę poszczególnych anomalii od momentu ich wykrycia do momentu zamknięcia zgłoszenia oraz reguły klasyfikacji takich anomalii. Na powyższy przepływ pracy składają się czynności związane z rejestrowaniem, analizowaniem i klasyfikowaniem zgłaszanych anomalii, podejmowaniem decyzji o właściwym sposobie reagowania (usunięcie problemu, pozostawienie bez zmian) oraz zamykaniem raportów o defektach. Ustalonej procedury muszą przestrzegać wszyscy zaangażowani interesariusze. Podobny sposób postępowania jest również zalecany w przypadku defektów wykrytych w trakcie testowania statycznego (a zwłaszcza analizy statycznej).

Typowy raport o defekcie ma na celu:

- dostarczenie osobom, które są odpowiedzialne za obsługę i usuwanie zgłoszonych defektów, informacji wystarczających do rozwiązania problemu;
- umożliwienie śledzenia jakości produktu pracy;
- przedstawienie sugestii dotyczących usprawnienia procesu wytwarzania oprogramowania i procesu testowego.

Raport o defekcie rejestrowany podczas testowania dynamicznego zawiera zwykle następujące informacje:

- jednoznaczny identyfikator;
- tytuł i krótkie podsumowanie zgłaszanej anomalii;

- data zaobserwowania anomalii, zgłaszająca jednostka organizacyjna i autor zgłoszenia (w tym jego rola);
- identyfikacja przedmiotu testów i środowiska testowego;
- kontekst wystąpienia defektu (np. uruchamiany przypadek testowy, wykonywana czynność testowa, faza cyklu wytwarzania oprogramowania oraz inne istotne informacje, takie jak stosowana technika testowania, lista kontrolna czy dane testowe);
- opis awarii umożliwiający jej odtworzenie i usunięcie (w tym kroki, które umożliwiły wykrycie anomalii) oraz wszelkie istotne dzienniki testów, zrzuty baz danych, zrzuty ekranu lub nagrania;
- oczekiwane i rzeczywiste rezultaty;
- krytyczność (stopień wpływu) defektu z punktu widzenia interesariuszy lub wymagań;
- priorytet usunięcia;
- status defektu (np. otwarty, odroczony, powielony, oczekujący na poprawkę, oczekujący na testowanie potwierdzające, ponownie otwarty, zamknięty, odrzucony);
- odwołania do innych elementów (np. do przypadku testowego).

Niektóre z powyższych danych (np. identyfikator, data, autor i początkowy status) mogą zostać uwzględnione automatycznie w przypadku korzystania z narzędzi do zarządzania defektami. Szablony raportów o defektach i przykładowe raporty tego typu przedstawiono w standardzie ISO/IEC/IEEE 29119-3 (raporty o defektach są w nim nazywane raportami o incydentach).

# 6. Narzędzia testowe — 20 minut

**Słowa kluczowe**

automatyzacja testów

**Cele nauczania w rozdziale 6:**

### 6.1 Narzędzia wspomagające testowanie

FL-6.1.1 (K2) Kandydat wyjaśnia, w jaki sposób różnego typu narzędzia testowe wspomagają testowanie.

### 6.2 Korzyści i ryzyka związane z automatyzacją testowania

FL-6.2.1 (K1) Kandydat pamięta korzyści i ryzyka związane z automatyzacją testowania.

## 6.1. Narzędzia wspomagające testowanie

Narzędzia testowe wspomagają i ułatwiają wykonywanie wielu czynności testowych. Dostępne są między innymi następujące narzędzia:

- narzędzia do zarządzania — zwiększające efektywność procesu testowego poprzez ułatwienie zarządzania cyklem wytwarzania oprogramowania, wymaganiami, testami, defektami i konfiguracją;
- narzędzia do testowania statycznego — pomagające testerom w przeprowadzaniu przeglądów i analizy statycznej;
- narzędzia do projektowania i implementacji testów — ułatwiające tworzenie przypadków, danych i procedur testowych;
- narzędzia do wykonywania testów i pomiaru pokrycia — ułatwiające automatyczne wykonywanie testów i mierzenie pokrycia;
- narzędzia do testowania niefunkcjonalnego — umożliwiające testerom wykonywanie testów niefunkcjonalnych, które są trudne lub niemożliwe do wykonania w trybie manualnym;
- narzędzia DevOps — wspomagające działanie potoku dostarczania, śledzenie przepływu pracy, wykonywanie zautomatyzowanych procesów budowania oraz funkcjonowanie mechanizmów ciągłej integracji i ciągłego dostarczania w metodyce DevOps;
- narzędzia wspomagające współpracę — ułatwiające wymianę informacji;
- narzędzia zwiększające skalowalność i standaryzację wdrażania (np. maszyny wirtualne lub narzędzia do konteneryzacji);
- wszelkie inne narzędzia wspomagające testowanie (np. w kontekście testowania narzędziem testowym może być również arkusz kalkulacyjny).

## 6.2. Korzyści i ryzyka związane z automatyzacją testów

Sam zakup narzędzia nie gwarantuje jeszcze sukcesu. Osiągnięcie realnych i trwałych korzyści z wdrożenia nowego narzędzia zawsze wymaga dodatkowego wysiłku (związanego np. z wprowadzeniem i utrzymaniem takiego narzędzia oraz przeprowadzeniem związanych z nim szkoleń). Ponadto należy również uwzględnić pewne ryzyka, które wymagają przeanalizowania i złagodzenia.

Potencjalne korzyści wynikające z automatyzacji testów to między innymi:

- oszczędność czasu poprzez ograniczenie powtarzalnych czynności wykonywanych manualnie (takich jak uruchamianie testów regresji, wielokrotne wprowadzanie tych samych danych testowych, porównywanie rzeczywistych rezultatów z oczekiwanymi czy sprawdzanie zgodności ze standardami tworzenia kodu);
- zapobieganie prostym błędom ludzkim poprzez zwiększenie spójności i powtarzalności (np. poprzez wyprowadzanie testów w spójny sposób z wymagań, systematyczne tworzenie danych testowych oraz wykonywanie testów przy użyciu narzędzia w tej samej kolejności i z tą samą częstotliwością);

- bardziej obiektywna ocena (np. w zakresie pokrycia) i przeprowadzanie pomiarów, które są zbyt skomplikowane, aby mogły zostać dokonane przez ludzi;
- łatwiejszy dostęp do informacji na temat testowania, które ułatwiają zarządzanie testami i raportowanie na temat testów (np. danych statystycznych, wykresów i zagregowanych danych obrazujących postęp testów, współczynników występowania defektów oraz danych dotyczących czasu trwania wykonywanych testów);
- skrócenie czasu wykonywania testów, co przekłada się na wcześniejsze wykrywanie defektów oraz szybsze przekazywanie informacji zwrotnych i wprowadzanie produktów na rynek;
- zapewnienie testerom dodatkowego czasu na projektowanie nowych, bardziej wnikliwych i skuteczniejszych testów.

Do potencjalnych ryzyk związanych z automatyzacją testów należą:

- nierealistyczne oczekiwania co do korzyści wynikających z zastosowania narzędzia (w tym funkcjonalności i łatwości obsługi);
- niedokładne oszacowanie czasu, kosztów i nakładów pracy związanych z wprowadzeniem narzędzia, utrzymaniem skryptów testowych oraz zmianą dotychczasowego manualnego procesu testowego;
- stosowanie narzędzia w sytuacjach, w których lepiej sprawdzi się testowanie manualne;
- nadmierne uzależnienie od narzędzia (np. lekceważenie potrzeby krytycznego myślenia);
- uzależnienie od dostawcy narzędzia, który może zakończyć działalność, wycofać narzędzie lub sprzedać je innemu dostawcy bądź świadczyć niskiej jakości usługi wsparcia technicznego (np. w zakresie reagowania na zapytania, dostarczania uaktualnień lub usuwania defektów);
- korzystanie z oprogramowania *open source*, które może zostać porzucone (co oznacza brak dalszych aktualizacji) lub którego elementy wewnętrzne mogą wymagać stosunkowo częstych aktualizacji w związku z dalszymi pracami rozwojowymi;
- brak kompatybilności narzędzia do automatyzacji z daną platformą programistyczną;
- wybór nieodpowiedniego narzędzia, które nie spełnia wymogów prawnych i/lub norm bezpieczeństwa.

# 7. Bibliografia

## Normy i standardy

ISO/IEC/IEEE 29119-1 (2022) Software and systems engineering — Software testing — Part 1: General Concepts

ISO/IEC/IEEE 29119-2 (2021) Software and systems engineering — Software testing — Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2021) Software and systems engineering — Software testing — Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2021) Software and systems engineering — Software testing — Part 4: Test techniques

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

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

ISO/IEC/IEEE 14764:2022 — Software engineering — Software life cycle processes — Maintenance

ISO 31000 (2018) Risk management — Principles and guidelines

## Książki

Adzic, G. (2009) *Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing*, Neuri Limited

Ammann, P., Offutt, J. (2016) *Introduction to Software Testing* (wyd. 2), Cambridge University Press

Andrews, M., Whittaker, J. (2006) *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*, Addison-Wesley Professional

Beck, K. (2003) *Test Driven Development: By Example*, Addison-Wesley

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

Boehm, B. (1981) *Software Engineering Economics*, Prentice Hall: Englewood Cliffs, NJ

Buxton, J. N., Randell B. (red.) (1970) *Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969*, s. 16

Chelimsky, D. i in. (2010) *The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends*, The Pragmatic Bookshelf: Raleigh, NC

Cohn, M. (2009) *Succeeding with Agile: Software Development Using Scrum*, Addison-Wesley

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

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

Crispin, L., Gregory, J. (2008) *Agile Testing: A Practical Guide for Testers and Agile Teams*, Pearson Education: Boston, MA

Forgács, I., Kovács, A. (2019) *Practical Test Design: Selection of traditional and automated test design techniques*, BCS, The Chartered Institute for IT

Gawande, A. (2009) *The Checklist Manifesto: How to Get Things Right*, Metropolitan Books: New York, NY

Gärtner, M. (2011) *ATDD by Example: A Practical Guide to Acceptance Test-Driven Development*, Pearson Education: Boston, MA

Gilb, T., Graham, D. (1993) *Software Inspection*, Addison-Wesley

Hendrickson, E. (2013) *Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing*, The Pragmatic Programmers

Hetzel, B. (1988) *The Complete Guide to Software Testing* (wyd. 2), John Wiley and Sons

Jeffries, R., Anderson, A., Hendrickson, C. (2000) *Extreme Programming Installed*, Addison-Wesley Professional

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

Kan, S. (2003) *Metrics and Models in Software Quality Engineering* (wyd. 2), Addison-Wesley, polskie wydanie: (2006) *Metryki i modele w inżynierii jakości oprogramowania*, Mikom

Kaner, C., Falk, J., Nguyen, H. Q. (1999) *Testing Computer Software* (wyd. 2), Wiley Computer Publishing

Kaner, C., Bach, J., Pettichord, B. (2011) *Lessons Learned in Software Testing: A Context-Driven Approach* (wyd. 1), Wiley Computer Publishing

Kim, G., Humble, J., Debois, P., Willis, J. (2016) *The DevOps Handbook*, IT Revolution Press: Portland, OR

Koomen, T., van der Aalst, L., Broekman, B., Vroon, M. (2006) *TMap Next for result-driven testing*, UTN Publishers

Myers, G. (2011) *The Art of Software Testing* (wyd. 3), John Wiley & Sons: New York, NY, polskie wydanie: (2005) *Sztuka testowania oprogramowania*, Helion

O'Regan, G. (2019) *Concise Guide to Software Testing*, Springer Nature Switzerland

Pressman, R. S. (2019) *Software Engineering. A Practitioner's Approach* (wyd. 9), McGraw Hill, polskie wydanie: (2010) *Praktyczne podejście do inżynierii oprogramowania*, WNT

Roman, A. (2018) *Thinking-Driven Testing. The Most Reasonable Approach to Quality Control*, Springer Nature Switzerland

Van Veenendaal, E. (red.) (2012) *Practical Risk-Based Testing: The PRISMA Approach*, UTN Publishers

Watson, A. H., Wallace, D. R., McCabe, T. J. (1996) *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*, U.S. Dept. of Commerce, Technology Administration, NIST

Westfall, L. (2009) *The Certified Software Quality Engineer Handbook*, ASQ Quality Press

Whittaker, J. (2002) *How to Break Software: A Practical Guide to Testing*, Pearson

Whittaker, J. (2009) *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design*, Addison-Wesley

Whittaker, J., Thompson, H. (2003) *How to Break Software Security*, Addison-Wesley

Wiegers, K. (2001) *Peer Reviews in Software: A Practical Guide*, Addison-Wesley Professional

## Artykuły i strony internetowe

Brykczynski, B. (1999) „A survey of software inspection checklists”, *ACM SIGSOFT Software Engineering Notes* 24(1), s. 82–89

Enders, A. (1975) „An Analysis of Errors and Their Causes in System Programs”, *IEEE Transactions on Software Engineering* 1(2), s. 140–149

Manna, Z., Waldinger, R. (1978) “The logic of computer programming”, *IEEE Transactions on Software Engineering* 4(3), s. 199–229

Marick, B. (2003) „Exploration through Example”, http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1

Nielsen, J. (1994) „Enhancing the explanatory power of usability heuristics”, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence*, ACM Press, s. 152–158

Salman. I. (2016) „Cognitive biases in software quality and testing”, *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16)*, ACM Press, s. 823–826

Wake, B. (2003) „INVEST in Good Stories, and SMART Tasks”, https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

# 8. Załącznik A. Cele nauczania i poziomy poznawcze

Poniżej przedstawiono cele nauczania obowiązujące w przypadku niniejszego sylabusa. Znajomość każdego z zagadnień poruszonych w sylabusie będzie sprawdzana na egzaminie zgodnie z przypisanym celem nauczania. Opis celu nauczania zawiera czasownik wyrażający czynność, który odpowiada właściwemu poziomowi wiedzy wymienionemu poniżej.

**Poziom 1 — zapamiętać (K1):** Kandydat pamięta, rozpoznaje lub umie sobie przypomnieć dany termin lub dane pojęcie.

**Czynności:** pamiętać, rozpoznać, określić, wskazać.

**Przykłady:**

- „Kandydat wskazuje typowe cele testów".
- „Kandydat pamięta pojęcia związane z piramidą testów".
- „Kandydat rozpoznaje, jaki jest wkład testera w planowanie iteracji i wydań".

**Poziom 2 — zrozumieć (K2):** Kandydat potrafi uzasadnić lub wyjaśnić stwierdzenia dotyczące danego zagadnienia, a także podsumować, porównać i sklasyfikować pojęcia z zakresu testowania oraz podać odpowiednie przykłady.

**Czynności:** sklasyfikować, porównać, zestawić ze sobą, objaśnić, wyjaśnić, omówić, rozróżnić, odróżnić, wyjaśnić, podać przykłady, podsumować.

**Przykłady:**

- „Kandydat klasyfikuje różne sposoby pisania kryteriów akceptacji".
- „Kandydat porównuje poszczególne role występujące w testowaniu".
- „Kandydat rozróżnia ryzyka projektowe i produktowe".
- „Kandydat omawia na przykładach cel i treść planu testów".
- „Kandydat wyjaśnia wpływ wybranego modelu cyklu wytwarzania oprogramowania na testowanie".
- „Kandydat podsumowuje czynności wykonywane w ramach procesu przeglądu".

**Poziom 3 — zastosować (K3):** Kandydat potrafi wykonać odpowiednią procedurę, gdy zostanie mu postawione znane mu zadanie, bądź wybrać właściwą procedurę i zastosować ją w danym kontekście.

**Czynności**: zastosować, obliczyć, sporządzić, użyć.

**Przykłady:**

- „Kandydat stosuje priorytetyzację przypadków testowych".
- „Kandydat sporządza raport o defekcie".
- „Kandydat używa techniki analizy wartości brzegowych, aby zaprojektować przypadki testowe".

**Materiały dodatkowe** w zakresie poziomów poznawczych celów nauczania:

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

# 9. Załącznik B. Macierz powiązań między celami biznesowymi a celami nauczania

W tym rozdziale wskazano numery celów nauczania na poziomie podstawowym związanych z celami biznesowymi oraz przedstawiono powiązania między celami biznesowymi a celami nauczania na poziomie podstawowym.

| Cele biznesowe — poziom podstawowy | | FL-BO 1 | FL-BO 2 | FL-BO 3 | FL-BO 4 | FL-BO 5 | FL-BO 6 | FL-BO 7 | FL-BO 8 | FL-BO 9 | FL-BO 10 | FL-BO 11 | FL-BO 12 | FL-BO 13 | FL-BO 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BO1 | Znajomość istoty testowania i wynikających z niego korzyści | 6 | | | | | | | | | | | | | |
| BO2 | Znajomość podstawowych pojęć związanych z testowaniem oprogramowania | | 22 | | | | | | | | | | | | |
| BO3 | Identyfikowanie podejścia do testowania i czynności testowych, które mają być realizowane w zależności od kontekstu testowania | | | 6 | | | | | | | | | | | |
| BO4 | Dokonywanie oceny i podnoszenie jakości dokumentacji | | | | 9 | | | | | | | | | | |
| BO5 | Zwiększanie skuteczności i efektywności testowania | | | | | 20 | | | | | | | | | |
| BO6 | Dopasowywanie procesu testowego do cyklu wytwarzania oprogramowania | | | | | | 6 | | | | | | | | |
| BO7 | Znajomość zasad zarządzania testami | | | | | | | 6 | | | | | | | |
| BO8 | Sporządzanie i udostępnianie przejrzystych, zrozumiałych raportów o defektach | | | | | | | | 1 | | | | | | |
| BO9 | Znajomość czynników wpływających na priorytety i pracochłonność testowania | | | | | | | | | 7 | | | | | |
| BO10 | Praca w zespole interdyscyplinarnym | | | | | | | | | | 8 | | | | |
| BO11 | Znajomość ryzyk i korzyści związanych z automatyzacją testów | | | | | | | | | | | 1 | | | |
| BO12 | Identyfikowanie niezbędnych umiejętności wymaganych w związku z testowaniem | | | | | | | | | | | | 5 | | |
| BO13 | Znajomość wpływu ryzyka na testowanie | | | | | | | | | | | | | 4 | |
| BO14 | Sprawne raportowanie na temat postępu i jakości testów | | | | | | | | | | | | | | 4 |

| Rozdział/ podrozdział/ sekcja | Cel nauczania | Poziom wiedzy („K”) | CELE BIZNESOWE | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | FL-BO1 | FL-BO2 | FL-BO3 | FL-BO4 | FL-BO5 | FL-BO6 | FL-BO7 | FL-BO8 | FL-BO9 | FL-BO10 | FL-BO11 | FL-BO12 | FL-BO13 | FL-BO14 |
| **Rozdział 1** | **Podstawy testowania** | | | | | | | | | | | | | | | |
| **1.1** | **Co to jest testowanie?** | | | | | | | | | | | | | | | |
| 1.1.1 | Kandydat wskazuje typowe cele testów. | K1 | X | | | | | | | | | | | | | |
| 1.1.2 | Kandydat odróżnia testowanie od debugowania. | K2 | | X | | | | | | | | | | | | |
| **1.2** | **Dlaczego testowanie jest niezbędne?** | | | | | | | | | | | | | | | |
| 1.2.1 | Kandydat podaje przykłady wskazujące, dlaczego testowanie jest niezbędne. | K2 | X | | | | | | | | | | | | | |
| 1.2.2 | Kandydat pamięta, jaka jest relacja między testowaniem a zapewnieniem jakości. | K1 | | X | | | | | | | | | | | | |
| 1.2.3 | Kandydat odróżnia podstawową przyczynę, pomyłkę, defekt i awarię. | K2 | | X | | | | | | | | | | | | |
| **1.3** | **Zasady testowania** | | | | | | | | | | | | | | | |
| 1.3.1 | Kandydat objaśnia siedem zasad testowania. | K2 | | X | | | | | | | | | | | | |
| **1.4** | **Czynności testowe, testalia i role związane z testami** | | | | | | | | | | | | | | | |
| 1.4.1 | Kandydat podsumowuje poszczególne czynności i zadania testowe. | K2 | | | X | | | | | | | | | | | |
| 1.4.2 | Kandydat wyjaśnia wpływ kontekstu na proces testowy. | K2 | | | X | | | X | | | | | | | | |
| 1.4.3 | Kandydat rozróżnia testalia wspomagające czynności testowe. | K2 | | | X | | | | | | | | | | | |
| 1.4.4 | Kandydat wyjaśnia korzyści wynikające ze śledzenia powiązań. | K2 | | | | X | X | | | | | | | | | |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.4.5 | Kandydat porównuje poszczególne role występujące w testowaniu. | K2 | | | | | | | | | | X | | | | |
| **1.5** | **Niezbędne umiejętności i dobre praktyki w dziedzinie testowania** | | | | | | | | | | | | | | | |
| 1.5.1 | Kandydat podaje przykłady ogólnych umiejętności wymaganych w związku z testowaniem. | K2 | | | | | | | | | | | | X | | |
| 1.5.2 | Kandydat pamięta, jakie są zalety podejścia opartego na zaangażowaniu całego zespołu. | K1 | | | | | | | | | | X | | | | |
| 1.5.3 | Kandydat wskazuje zalety i wady niezależności testowania. | K2 | | | X | | | | | | | | | | | |
| **Rozdział 2** | **Testowanie w cyklu wytwarzania oprogramowania** | | | | | | | | | | | | | | | |
| **2.1** | **Testowanie w kontekście cyklu wytwarzania oprogramowania** | | | | | | | | | | | | | | | |
| 2.1.1 | Kandydat wyjaśnia wpływ wybranego modelu cyklu wytwarzania oprogramowania na testowanie. | K2 | | | | | | X | | | | | | | | |
| 2.1.2 | Kandydat pamięta dobre praktyki testowania mające zastosowanie do wszystkich cykli wytwarzania oprogramowania. | K1 | | | | | | X | | | | | | | | |
| 2.1.3 | Kandydat podaje przykłady podejść typu „najpierw test” w kontekście wytwarzania oprogramowania. | K1 | | | | | X | | | | | | | | | |
| 2.1.4 | Kandydat podsumowuje, w jaki sposób metodyka DevOps może wpłynąć na testowanie. | K2 | | | | | X | X | | | X | X | | | | |
| 2.1.5 | Kandydat wyjaśnia, na czym polega przesunięcie w lewo. | K2 | | | | | X | X | | | | | | | | |
| 2.1.6 | Kandydat wyjaśnia, w jaki sposób retrospektywy mogą posłużyć do doskonalenia procesów. | K2 | | | | | X | | | | | X | | | | |
| **2.2** | **Poziomy testów i typy testów** | | | | | | | | | | | | | | | |
| 2.2.1 | Kandydat rozróżnia poszczególne poziomy testów. | K2 | | X | X | | | | | | | | | | | |
| 2.2.2 | Kandydat rozróżnia poszczególne typy testów. | K2 | | X | | | | | | | | | | | | |
| 2.2.3 | Kandydat odróżnia testowanie potwierdzające od testowania regresji. | K2 | | X | | | | | | | | | | | | |
| **2.3** | **Testowanie pielęgnacyjne** | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2.3.1 | Kandydat podsumowuje testowanie pielęgnacyjne i zdarzenia wyzwalające ten rodzaj testowania. | K2 | | X | | | | | X | | | | | | | |
| **Rozdział 3** | **Testowanie statyczne** | | | | | | | | | | | | | | | |
| **3.1** | **Podstawy testowania statycznego** | | | | | | | | | | | | | | | |
| 3.1.1 | Kandydat rozpoznaje typy produktów, które mogą być badane przy użyciu poszczególnych technik testowania statycznego. | K1 | | | | X | X | | | | | | | | | |
| 3.1.2 | Kandydat wyjaśnia korzyści wynikające z testowania statycznego. | K2 | X | | | X | X | | | | | | | | | |
| 3.1.3 | Kandydat porównuje i zestawia ze sobą testowanie statyczne i dynamiczne. | K2 | | | | X | X | | | | | | | | | |
| **3.2** | **Informacje zwrotne i proces przeglądu** | | | | | | | | | | | | | | | |
| 3.2.1 | Kandydat pamięta korzyści wynikające z wczesnego i częstego otrzymywania informacji zwrotnych od interesariuszy. | K1 | X | | | X | | | | | | X | | | | |
| 3.2.2 | Kandydat podsumowuje czynności wykonywane w ramach procesu przeglądu. | K2 | | | X | X | | | | | | | | | | |
| 3.2.3 | Kandydat pamięta, jakie obowiązki są przypisane do najważniejszych ról w trakcie wykonywania przeglądów. | K1 | | | | X | | | | | | | | X | | |
| 3.2.4 | Kandydat porównuje i zestawia ze sobą różne typy przeglądów. | K2 | | X | | | | | | | | | | | | |
| 3.2.5 | Kandydat pamięta, jakie czynniki decydują o powodzeniu przeglądu. | K1 | | | | | X | | | | | | | X | | |
| **Rozdział 4** | **Analiza i projektowanie testów** | | | | | | | | | | | | | | | |
| **4.1** | **Ogólna charakterystyka technik testowania** | | | | | | | | | | | | | | | |
| 4.1.1 | Kandydat rozróżnia czarnoskrzynkowe i białoskrzynkowe techniki testowania oraz techniki testowania oparte na doświadczeniu. | K2 | | X | | | | | | | | | | | | |
| **4.2** | **Czarnoskrzynkowe techniki testowania** | | | | | | | | | | | | | | | |
| 4.2.1 | Kandydat używa techniki podziału na klasy równoważności, aby zaprojektować przypadki testowe. | K3 | | | | | X | | | | | | | | | |
| 4.2.2 | Kandydat używa techniki analizy wartości brzegowych, aby zaprojektować przypadki testowe. | K3 | | | | | X | | | | | | | | | |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4.2.3 | Kandydat używa techniki testowania w oparciu o tablicę decyzyjną, aby zaprojektować przypadki testowe. | K3 | | | | | X | | | | | | | | | |
| 4.2.4 | Kandydat używa techniki testowania przejść pomiędzy stanami, aby zaprojektować przypadki testowe. | K3 | | | | | X | | | | | | | | | |
| **4.3** | **Białoskrzynkowe techniki testowania** | | | | | | | | | | | | | | | |
| 4.3.1 | Kandydat wyjaśnia pojęcie testowanie instrukcji. | K2 | | X | | | | | | | | | | | | |
| 4.3.2 | Kandydat wyjaśnia pojęcie testowanie gałęzi. | K2 | | X | | | | | | | | | | | | |
| 4.3.3 | Kandydat wyjaśnia korzyści wynikające z testowania białoskrzynkowego. | K2 | X | X | | | | | | | | | | | | |
| **4.4** | **Techniki testowania oparte na doświadczeniu** | | | | | | | | | | | | | | | |
| 4.4.1 | Kandydat wyjaśnia pojęcie zgadywanie błędów. | K2 | | X | | | | | | | | | | | | |
| 4.4.2 | Kandydat wyjaśnia pojęcie testowanie eksploracyjne. | K2 | | X | | | | | | | | | | | | |
| 4.4.3 | Kandydat wyjaśnia pojęcie testowanie w oparciu o listę kontrolną. | K2 | | X | | | | | | | | | | | | |
| **4.5** | **Podejścia do testowania oparte na współpracy** | | | | | | | | | | | | | | | |
| 4.5.1 | Kandydat wyjaśnia, w jaki sposób należy pisać historyjki użytkownika we współpracy z programistami i przedstawicielami jednostek biznesowych. | K2 | | | | X | | | | | | X | | | | |
| 4.5.2 | Kandydat klasyfikuje różne sposoby pisania kryteriów akceptacji. | K2 | | | | | | | | | | X | | | | |
| 4.5.3 | Kandydat projektuje przypadki testowe metodą wytwarzania sterowanego testami akceptacyjnymi. | K3 | | | | | X | | | | | | | | | |
| **Rozdział 5** | **Zarządzanie czynnościami testowymi** | | | | | | | | | | | | | | | |
| **5.1** | **Planowanie testów** | | | | | | | | | | | | | | | |
| 5.1.1 | Kandydat omawia na przykładach cel i treść planu testów. | K2 | | X | | | | | X | | | | | | | |
| 5.1.2 | Kandydat rozpoznaje, jaki jest wkład testera w planowanie iteracji i wydań. | K1 | X | | | | | | | | | X | | X | | |
| 5.1.3 | Kandydat porównuje i zestawia ze sobą kryteria wejścia i kryteria wyjścia. | K2 | | | | X | | X | | | | | | | | X |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5.1.4 | Kandydat oblicza pracochłonność testowania przy użyciu technik szacowania. | K3 | | | | | | | X | | X | | | | | |
| 5.1.5 | Kandydat stosuje priorytetyzację przypadków testowych. | K3 | | | | | | | X | | X | | | | | |
| 5.1.6 | Kandydat pamięta pojęcia związane z piramidą testów. | K1 | | X | | | | | | | | | | | | |
| 5.1.7 | Kandydat podsumowuje kwadranty testowe oraz ich relację do poziomów testów i typów testów. | K2 | | X | | | | | | | X | | | | | |
| **5.2** | **Zarządzanie ryzykiem** | | | | | | | | | | | | | | | |
| 5.2.1 | Kandydat określa poziom ryzyka na podstawie prawdopodobieństwa ryzyka i wpływu ryzyka. | K1 | | | | | | | X | | | | | | X | |
| 5.2.2 | Kandydat rozróżnia ryzyka projektowe i produktowe. | K2 | | X | | | | | | | | | | | X | |
| 5.2.3 | Kandydat wyjaśnia potencjalny wpływ analizy ryzyka produktowego na staranność i zakres testowania. | K2 | | | | | X | | | | X | | | | X | |
| 5.2.4 | Kandydat wyjaśnia, jakie środki można podjąć w odpowiedzi na przeanalizowane ryzyka produktowe. | K2 | | X | | | X | | | | | | | | X | |
| **5.3** | **Monitorowanie testów, nadzór nad testami i ukończenie testów** | | | | | | | | | | | | | | | |
| 5.3.1 | Kandydat pamięta metryki stosowane w odniesieniu do testowania. | K1 | | | | | | | | | X | | | | | X |
| 5.3.2 | Kandydat podsumowuje cele i treść raportów z testów oraz wskazuje ich odbiorców. | K2 | | | | | X | | | | X | | | | | X |
| 5.3.3 | Kandydat omawia na przykładach sposób przekazywania informacji o statusie testowania. | K2 | | | | | | | | | | | | X | | X |
| **5.4** | **Zarządzanie konfiguracją** | | | | | | | | | | | | | | | |
| 5.4.1 | Kandydat podsumowuje, w jaki sposób zarządzanie konfiguracją wspomaga testowanie. | K2 | | | | | X | | X | | | | | | | |
| **5.5** | **Zarządzanie defektami** | | | | | | | | | | | | | | | |
| 5.5.1 | Kandydat sporządza raport o defekcie. | K3 | | X | | | | | | X | | | | | | |
| **Rozdział 6** | **Narzędzia testowe** | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **6.1** | **Narzędzia wspomagające testowanie** | | | | | | | | | | | | | | | |
| 6.1.1 | Kandydat wyjaśnia, w jaki sposób różnego typu narzędzia testowe wspomagają testowanie. | K2 | | | | | X | | | | | | | | | |
| **6.2** | **Korzyści i ryzyka związane z automatyzacją testowania** | | | | | | | | | | | | | | | |
| 6.2.1 | Kandydat pamięta korzyści i ryzyka związane z automatyzacją testowania. | K1 | | | | | X | | | | | | X | | | |

# 10. Załącznik C. Opis wydania

Wersja 4.0 stanowi istotną aktualizację sylabusa poziomu podstawowego ISTQB® opracowaną na podstawie sylabusa poziomu podstawowego w wersji 3.1.1 oraz sylabusa dla testerów zwinnych w wersji 2014. W związku z tym nie sporządzono szczegółowego opisu wydania w podziale na rozdziały i podrozdziały, a jedynie przedstawiono podsumowanie najważniejszych zmian. Ponadto w oddzielnym dokumencie z opisem wydania przedstawiono powiązania między celami nauczania sylabusa poziomu podstawowego w wersji 3.1.1 i sylabusa dla testerów zwinnych w wersji 2014 a celami nauczania w nowym sylabusie poziomu podstawowego w wersji 4.0 (ze wskazaniem celów dodanych, zaktualizowanych lub usuniętych).

Do momentu powstania niniejszego sylabusa (tj. do przełomu lat 2022 i 2023) do egzaminu na poziomie podstawowym przystąpiło ponad milion osób z ponad 100 krajów, a liczba certyfikowanych testerów na całym świecie przekroczyła 800 tys. Przy założeniu, że wszystkie te osoby przeczytały sylabus poziomu podstawowego, aby zdać egzamin, dokument ten jest prawdopodobnie najczęściej czytanym opracowaniem dotyczącym testowania oprogramowania! Obecna, gruntownie zaktualizowana wersja czerpie z tego dziedzictwa, a jednocześnie pozwala jeszcze lepiej zaprezentować jakość usług, które ISTQB® oferuje globalnej społeczności testerów, kolejnym setkom tysięcy odbiorców.

W bieżącej wersji wszystkie cele nauczania zostały przeredagowane w taki sposób, aby każdy z nich stanowił niepodzielną całość oraz aby można było jednoznacznie powiązać cele nauczania z treścią podrozdziałów sylabusa. Zadbano też o to, aby sylabus nie zawierał treści niepowiązanych z żadnymi celami nauczania. Autorzy skupili się na możliwości praktycznego wykorzystania przedstawionych treści oraz na równowadze między wiedzą a umiejętnościami, dzięki czemu nowa wersja powinna być bardziej przystępna i zrozumiała oraz łatwiejsza do przetłumaczenia, a zawarty w niej materiał — łatwiejszy do opanowania.

W bieżącym wydaniu głównym wprowadzono następujące zmiany:

- Zmniejszono objętość całego sylabusa. Sylabus nie jest podręcznikiem, lecz dokumentem mającym na celu przedstawienie w skrócie podstawowych elementów kursu wprowadzającego do tematyki testowania oprogramowania, w tym wskazanie, jakie tematy należy poruszyć i na jakim poziomie. W szczególności:
    - z tekstu usunięto w większości przypadków przykłady, ponieważ przygotowanie przykładów i ćwiczeń do wykorzystania w trakcie kursu jest zadaniem dostawcy szkoleń;
    - zastosowano „listę kontrolną pisania sylabusów", która określa sugerowaną maksymalną objętość tekstu (w j. angielskim) poszczególnych celów nauczania na poszczególnych poziomach wiedzy (K1 = maks. 10 wierszy, K2 = maks. 15 wierszy, K3 = maks. 25 wierszy).
- Zmniejszono liczbę celów nauczania (LO - *Learning Objectives*) w porównaniu z sylabusem poziomu podstawowego w wersji 3.1.1 oraz sylabusem dla testerów zwinnych w wersji 2014:
    - 14 LO na poziomie K1 w porównaniu z 21 LO w sylabusie poziomu podstawowego w wersji 3.1.1 (15) i sylabusie dla testerów zwinnych w wersji 2014 (6);
    - 42 LO na poziomie K2 w porównaniu z 53 LO w sylabusie poziomu podstawowego w wersji 3.1.1 (40) i sylabusie dla testerów zwinnych w wersji 2014 (13);
    - 8 LO na poziomie K3 w porównaniu z 15 LO w sylabusie poziomu podstawowego w wersji 3.1.1 (7) i sylabusie dla testerów zwinnych w wersji 2014 (8).

- Zwiększono liczbę odwołań do klasycznych i/lub powszechnie uznanych książek i artykułów na temat testowania oprogramowania i zagadnień pokrewnych.
- Istotne zmiany w rozdziale 1 (Podstawy testowania):
  - Rozszerzono i poprawiono podrozdział dotyczący umiejętności w dziedzinie testowania.
  - Dodano sekcję dotyczącą podejścia opartego na zaangażowaniu całego zespołu (K1).
  - Przeniesiono sekcję dotyczącą niezależności testowania z rozdziału 5 do rozdziału 1.
- Istotne zmiany w rozdziale 2 (Testowanie w cyklu wytwarzania oprogramowania):
  - Przeredagowano i poprawiono treść sekcji 2.1.1 i 2.1.2 oraz zmodyfikowano odpowiadające im LO.
  - Poświęcono więcej uwagi praktykom takim jak podejście „najpierw test" (K1), przesunięcie w lewo (K2) czy retrospektywa (K2).
  - Dodano nową sekcję dotyczącą testowania w kontekście metodyki DevOps (K2).
  - Podzielono testowanie integracyjne na dwa oddzielne poziomy testów: testowanie integracji modułów i testowanie integracji systemów.
- Istotne zmiany w rozdziale 3 (Testowanie statyczne):
  - Usunięto sekcję dotyczącą technik przeglądu wraz z LO na poziomie K3 (stosowanie technik przeglądu).
- Istotne zmiany w rozdziale 4 (Analiza i projektowanie testów):
  - Usunięto treść dotyczącą testowania opartego na przypadkach użycia (treść ta jest nadal dostępna w sylabusie dla analityków testów na poziomie zaawansowanym).
  - Poświęcono więcej uwagi podejściu opartemu na współpracy poprzez dodanie nowego LO na poziomie K3 dotyczącego projektowania przypadków testowych metodą ATDD oraz dwóch nowych LO na poziomie K2 dotyczących historyjek użytkownika i kryteriów akceptacji.
  - Materiał dotyczący testowania i pokrycia decyzji zastąpiono materiałem dotyczącym testowania i pokrycia gałęzi. Wynika to z faktu, że: po pierwsze, pokrycie gałęzi jest częściej stosowane w praktyce; po drugie, różne standardy różnie definiują pojęcie „decyzji" (inaczej niż w przypadku „gałęzi"); po trzecie, pozwala to usunąć subtelną, ale poważną wadę występującą w dotychczasowym sylabusie poziomu podstawowego z 2018 r., w którym napisano, że „uzyskanie stuprocentowego pokrycia decyzji gwarantuje stuprocentowe pokrycie instrukcji kodu", co nie jest prawdą w przypadku programów, w których nie występują decyzje.
  - Poprawiono podrozdział dotyczący korzyści wynikających z testowania białoskrzynkowego.
- Istotne zmiany w rozdziale 5 (Zarządzanie czynnościami testowymi):
  - Usunięto sekcję dotyczącą strategii testów i podejść do testowania.
  - Dodano nowy LO na poziomie K3 dotyczący technik szacowania pracochłonności testowania.

- Omówiono szerzej dobrze znane pojęcia i narzędzia związane ze zwinnym wytwarzaniem oprogramowania w kontekście zarządzania testami: planowanie iteracji i wydań (K1), piramidę testów (K1) i kwadranty testowe (K2).
- Poprawiono strukturę podrozdziału dotyczącego zarządzania ryzykiem poprzez opisanie czterech głównych czynności, czyli: identyfikacji ryzyka, oceny ryzyka, łagodzenia ryzyka i monitorowania ryzyka.

- Istotne zmiany w rozdziale 6 (Narzędzia testowe):
  - Okrojono materiał dotyczący niektórych kwestii związanych z automatyzacją testów, który był zbyt zaawansowany jak na poziom podstawowy, w tym usunięto sekcje dotyczące wyboru narzędzi, przeprowadzania projektów pilotażowych i wprowadzania narzędzi w organizacji.

# 11. Indeks

## L



## Ł



## M



## N



## O



## P



## R

## S



## Ś



## T



## U



## W

## Z



## Ż