WYKŁAD 2

Spis treści

2.1. Modele cyklu życia oprogramowania

Model cyklu życia oprogramowania opisuje rodzaje czynności wykonywanych na poszczególnych etapach projektu wytwarzania oprogramowania oraz powiązania logiczne i chronologiczne między tymi czynnościami. Istnieje wiele różnych modeli cyklu życia oprogramowania, a każdy z nich wymaga innego podejścia do testowania.

2.1.1. Wytwarzanie oprogramowania a testowanie oprogramowania

Znajomość powszechnie stosowanych modeli cyklu życia oprogramowania jest ważnym elementem obowiązków każdego testera, ponieważ pozwala uwzględnić właściwe czynności testowe.

Poniżej wymieniono kilka zasad dobrych praktyk testowania, które dotyczą każdego modelu cyklu życia oprogramowania:

- dla każdej czynności związanej z wytwarzaniem oprogramowania istnieje odpowiadającą jej czynność testowa;
- każdy poziom testów ma przypisane cele odpowiednie do tego poziomu;
- analizę i projektowanie testów na potrzeby danego poziomu testów należy rozpocząć podczas wykonywania odpowiadającej danemu poziomowi czynności związanej zwytwarzaniem oprogramowania;
- testerzy powinni uczestniczyć w dyskusjach dotyczących definiowania i doprecyzowywania wymagań i założeń projektu oraz w przeglądach produktów pracy (np. wymagań, założeń projektu czy też historyjek użytkownika) natychmiast po udostępnieniu wersji roboczych odpowiednich dokumentów.

Bez względu na wybrany model cyklu życia oprogramowania czynności testowe powinny rozpocząć się w początkowym etapie tego cyklu, zgodnie z zasadą wczesnego testowania. W niniejszym sylabusie modele cyklu życia oprogramowania podzielono na następujące kategorie:

- sekwencyjne modele wytwarzania oprogramowania;
- iteracyjne i przyrostowe modele wytwarzania oprogramowania.

Sekwencyjny model wytwarzania oprogramowania opisuje proces wytwarzania oprogramowania jako liniowy przepływ czynności następujących kolejno po sobie. Oznacza to, że każda faza tego procesu powinna rozpoczynać się po zakończeniu fazy poprzedniej. W teorii poszczególne fazy nie zachodzą na siebie, ale w praktyce korzystne jest uzyskiwanie wczesnych informacji zwrotnych z kolejnej fazy.

W modelu kaskadowym (ang. *waterfall*) czynności związane z wytwarzaniem oprogramowania

(takie jak: analiza wymagań, projektowanie, tworzenie kodu czy testowanie) wykonuje się jedna po drugiej. Zgodnie z założeniami tego modelu czynności testowe występują dopiero wtedy, gdy wszystkie inne czynności wytwórcze zostaną ukończone.

W przeciwieństwie do modelu kaskadowego model V zakłada integrację procesu testowania z całym procesem wytwarzania oprogramowania, a tym samym wprowadza w życie zasadę wczesnego testowania. Ponadto model V obejmuje poziomy testowania powiązane z poszczególnymi, odpowiadającymi im, fazami wytwarzania oprogramowania, co dodatkowo sprzyja wczesnemu testowaniu (patrz podrozdział 2.2.). W tym modelu wykonanie testów powiązanych ze wszystkimi poziomami testów następuje sekwencyjnie, jednak w niektórych przypadkach może następować nachodzenie faz na siebie nawzajem.

W większości przypadków sekwencyjne modele wytwarzania oprogramowania pozwalają na wytworzenie oprogramowania posiadającego pełny zestaw funkcjonalności. Stosowanie takich modeli wytwarzania oprogramowania zwykle wymaga miesięcy, jeśli nie lat, pracy, aby zostało ono dostarczone do interesariuszy i użytkowników.

Przyrostowe wytwarzanie oprogramowania to proces polegający na określaniu wymagań oraz projektowaniu, budowaniu i testowaniu systemu częściami, co oznacza, że funkcjonalność oprogramowania rośnie przyrostowo. Wielkość poszczególnych przyrostów funkcjonalności zależy od konkretnego modelu cyklu życia: niektóre modele przewidują podział na większe fragmenty, a inne — na mniejsze. Jednorazowy przyrost funkcjonalności może ograniczać się nawet do wprowadzenia pojedynczej zmiany na ekranie interfejsu użytkownika lub nowej opcji zapytania.

Wytwarzanie iteracyjne polega na specyfikowaniu, projektowaniu, budowaniu i testowaniu wspólnie grup funkcjonalności w serii cykli, często o określonym czasie trwania. Iteracje mogą zawierać zmiany funkcjonalności wytworzonych we wcześniejszych iteracjach, wspólnie ze zmianami w zakresie projektu. Każda iteracja dostarcza działające oprogramowanie, które stanowi rosnący podzbiór całkowitego zbioru funkcjonalności aż do momentu, w którym końcowa wersja oprogramowania zostaje dostarczona lub wytwarzanie zostanie zatrzymane.

Wśród przykładów modeli iteracyjnych można wyróżnić:

- **Rational Unified Process** (RUP®): poszczególne iteracje trwają zwykle stosunkowo długo (np. dwa lub trzy miesiące), a przyrostowe części systemu są odpowiednio duże, czyli obejmują np. dwie lub trzy grupy powiązanych funkcjonalności.

- **SCRUM**: poszczególne iteracje trwają stosunkowo krótko (np. kilka godzin, dni lub tygodni), aprzyrostowe części systemu są odpowiednio małe, czyli obejmują na przykład dwa udoskonalenia i dwie lub trzy nowe funkcjonalności.

- **Kanban**: umożliwia dostarczenie jednego udoskonalenia lub jednej funkcjonalności naraz (natychmiast po przygotowaniu) bądź zgrupowanie większej liczby funkcjonalności w

celu równoczesnego przekazania na środowisko produkcyjne.

- **Model spiralny** (lub prototypowanie): tworzy się eksperymentalne elementy przyrostowe, które następnie mogą zostać gruntownie przebudowane, a nawet porzucone na dalszych etapach wytwarzania oprogramowania.

Komponenty systemów lub systemy wykorzystujące powyższe modele często odnoszą się do nachodzących na siebie i powtarzanych w cyklu życia oprogramowania poziomów testów. W idealnej sytuacji każda funkcjonalność jest testowana na wielu poziomach, zbliżając się do finalnego wydania. Czasami też zespoły korzystają z metody ciągłego dostarczania (ang. *Continuous Delivery*) lub ciągłego wdrażania oprogramowania. Te podejścia pozwalają na szerokie wykorzystywanie automatyzacji na wielu poziomach testowania (jako część potoku dostarczania oprogramowania). Ponadto wiele tego typu modeli uwzględnia koncepcję samoorganizujących się zespołów, która może zmienić sposób organizacji pracy w związku z testowaniem oraz relacje między testerami a programistami.

W każdym z powyższych modeli powstaje rosnący system, który może być dostarczany użytkownikom końcowym na zasadzie dodawania pojedynczych funkcjonalności, w określonych iteracjach lub w bardziej tradycyjny sposób, w formie dużych wydań. Niezależnie od tego, czy kolejne części przyrostowe oprogramowania są udostępniane użytkownikom, w miarę rozrastania się systemu coraz większego znaczenia nabiera testowanie regresji.

W przeciwieństwie do modeli sekwencyjnych, w modelach iteracyjno-przyrostowych działające oprogramowanie może być dostarczone już w ciągu tygodni, a nawet dni, jednak na spełnienie wszystkich wymagań potrzeba miesięcy, a nawet lat.

2.1.2. Modele cyklu życia oprogramowania w kontekście

Modele cyklu życia oprogramowania należy dobierać i dopasowywać do kontekstu wynikającego z charakterystyki projektu i produktu. W związku z tym w procesie wyboru i dostosowania odpowiedniego do potrzeb projektu modelu należy wziąć pod uwagę: cel projektu, typ wytwarzanego produktu, priorytety biznesowe (takie jak czas wprowadzenia produktu na rynek) i zidentyfikowane czynniki ryzyka produktowego i projektowego. Przykładem może być wytwarzanie i testowanie wewnętrznego systemu administracyjnego o niewielkim znaczeniu, które powinno przebiegać inaczej niż wytwarzanie i testowanie systemu krytycznego ze względów bezpieczeństwa takiego jak układ sterowania hamulcami w samochodzie. Innym przykładem może być sytuacja, w której kwestie organizacyjne i kulturowe mogą utrudniać komunikację pomiędzy członkami zespołu, co może skutkować spowolnieniem wytwarzania oprogramowania w modelu iteracyjnym.

W zależności od kontekstu projektu, konieczne może okazać się połączenie lub reorganizacja niektórych poziomów testów i/lub czynności testowych. W przypadku integracji oprogramowania do powszechnej sprzedaży (ang. *Commercial off-the shelf — COTS*) z większym systemem nabywca może wykonywać testy współdziałania na poziomie testowania integracji systemów (np. w zakresie integracji z infrastrukturą i innymi systemami) oraz na

poziomie testowania akceptacyjnego (testowanie funkcjonalne i niefunkcjonalne wraz z testowaniem akceptacyjnym przez użytkownika i produkcyjnymi testami akceptacyjnymi).

Różne modele cyklu życia oprogramowania można łączyć ze sobą. Przykładem może być zastosowanie **modelu V** do wytwarzania i testowania systemów back-endowych i ich integracji oraz modelu zwinnego do wytwarzania i testowania interfejsu użytkownika i funkcjonalności dostępnej dla użytkowników. Innym wariantem łączenia różnych modeli jest zastosowanie modelu prototypowania na wczesnym etapie projektu, a następnie zastąpienie go modelem przyrostowym po zakończeniu fazy eksperymentalnej.

W przypadku systemów związanych z Internetem rzeczy (ang. *Internet of Things — IoT*), które składają się z wielu różnych obiektów takich jak: urządzenia, produkty i usługi, w odniesieniu do poszczególnych obiektów stosuje się zwykle oddzielne modele cyklu życia. Stanowi to duże wyzwanie — zwłaszcza w zakresie wytwarzania poszczególnych wersji systemów IoT. Ponadto w przypadku powyższych obiektów większy nacisk kładzie się na późniejsze etapy cyklu życia oprogramowania, już po wprowadzeniu obiektów na produkcję (np. na fazy: produkcyjną, aktualizacji i wycofania z użytku).

Poniżej podano powody, dla których modele rozwoju oprogramowania muszą być dostosowane do kontekstu projektu i charakterystyki produktu:

- różnica w ryzykach produktowych systemów (projekt złożony lub prosty);
- wiele jednostek biznesowych może być częścią projektu lub programu (połączenie rozwoju sekwencyjnego i zwinnego);
- krótki czas na dostarczenie produktu na rynek (łączenie poziomów testów i/lub integracja typów testów na poziomach testowych).

2.2. Poziomy testów

Poziomy testów to grupy czynności testowych, które organizuje się i którymi zarządza się wspólnie. Każdy poziom testów jest instancją procesu testowego składającą się z czynności wykonywanych dla oprogramowania na danym poziomie wytwarzania, od pojedynczych modułów lub komponentów, po kompletne systemy lub, jeśli ma to miejsce w danym przypadku, systemy systemów. Poziomy te są powiązane z innymi czynnościami wykonywanymi w ramach cyklu życia oprogramowania.

Poziomy testów opisane w niniejszym sylabusie to:

- testowanie modułowe;
- testowanie integracyjne;
- testowanie systemowe;
- testowanie akceptacyjne.

Każdy poziom testów charakteryzują następujące atrybuty:

- cele szczegółowe;
- podstawa testów (do której należy odwoływać się przy wyprowadzaniu przypadków testowych);
- przedmiot testów (czyli to, co jest testowane);
- typowe defekty i awarie;
- konkretne podejścia i odpowiedzialności.

Każdy poziom testów wymaga odpowiedniego środowiska testowego, np. do testowania akceptacyjnego idealnie nadaje się środowisko testowe podobne do środowiska produkcyjnego, apodczas testowania modułowego programiści zwykle korzystają z własnego środowiska rozwojowego.

2.2.1. Testowanie modułowe

Cele testowania modułowego

Testowanie modułowe (zwane także testowaniem jednostkowym, testowaniem komponentów lub testowaniem programu) skupia się na modułach, które można przetestować oddzielnie. Cele tego typu testowania to między innymi:

- zmniejszanie ryzyka;
- sprawdzanie zgodności zachowań funkcjonalnych i niefunkcjonalnych modułu z projektem i specyfikacjami;
- budowanie zaufania do jakości modułu;
- wykrywanie defektów w module;
- zapobieganie przedostawaniu się defektów na wyższe poziomy testowania.

W niektórych przypadkach — zwłaszcza w przyrostowych i iteracyjnych modelach wytwarzania oprogramowania (np. modelu zwinnym), w których zmiany kodu mają charakter ciągły — automatyczne modułowe testy regresji są kluczowym elementem pozwalającym uzyskać pewność, że wprowadzone zmiany nie spowodowały nieprawidłowej pracy innych, istniejących i działających już modułów.

Testy modułowe są często wykonywane w izolacji od reszty systemu (zależnie od modelu cyklu życia oprogramowania i konkretnego systemu), przy czym w takiej sytuacji może być konieczne użycie atrap obiektów (ang. *mock object*), wirtualizacji usług, jarzm testowych, zaślepek bądź sterowników. Testowanie modułowe może obejmować funkcjonalność (np. poprawność obliczeń), parametry niefunkcjonalne (np. wycieki pamięci) oraz właściwości strukturalne (np.

testowanie decyzji).

Podstawa testów

Przykładowe produkty pracy, które mogą być wykorzystywane jako podstawa testów w ramach testowania modułowego, to między innymi:

- projekt szczegółowy;
- kod;
- model danych;
- specyfikacje modułów.

Przedmioty testów

Do typowych przedmiotów testów dla testów modułowych zalicza się:

- moduły, jednostki lub komponenty;
- kod i struktury danych;
- klasy;
- moduły baz danych.

Typowe defekty i awarie

Przykładami typowych defektów i awarii wykrywanych w ramach testowania modułowego są:

- niepoprawna funkcjonalność (np. niezgodna ze specyfikacją projektu);
- problemy z przepływem danych;
- niepoprawny kod i logika.

Defekty są zwykle usuwane natychmiast po wykryciu, często bez formalnego procesu zarządzania nimi. Należy jednak zaznaczyć, że zgłaszając defekty, programiści dostarczają ważne informacje na potrzeby analizy przyczyny podstawowej i doskonalenia procesów.

Konkretne podejścia i odpowiedzialności

Testowanie modułowe jest zwykle wykonywane przez programistę będącym autorem kodu, a przynajmniej wymaga dostępu do testowanego kodu. W związku z tym programiści mogą naprzemiennie tworzyć moduły i wykrywać/usuwać defekty. Programiści często piszą i wykonują testy po napisaniu kodu danego modułu. Jednakże, w niektórych przypadkach — zwłaszcza w przypadku zwinnego wytwarzania oprogramowania — automatyczne przypadki testowe do testowania modułowego można również tworzyć przed napisaniem kodu aplikacji.

Warto w tym miejscu omówić przykład wytwarzania sterowanego testami (ang. *Test Driven*

Development — TDD). Model ten ma charakter wysoce iteracyjny i opiera się na cyklach obejmujących tworzenie automatycznych przypadków testowych, budowanie i integrowanie niewielkich fragmentów kodu, wykonywanie testów modułowych, rozwiązywanie ewentualnych problemów oraz refaktoryzację kodu. Proces ten jest powtarzany do momentu ukończenia modułu i zaliczenia wszystkich testów modułowych. Wytwarzanie sterowane testami jest przykładem podejścia typu „najpierw testuj". Model ten był pierwotnie stosowany w ramach programowania ekstremalnego (ang. *eXtreme Programming — XP*), ale upowszechnił się również w innych modelach zwinnych oraz cyklach sekwencyjnych.

2.2.2. Testowanie integracyjne

Cele testowania integracyjnego

Testowanie integracyjne skupia się na interakcjach między modułami lub systemami. Cele testowania integracyjnego to:

- zmniejszanie ryzyka;
- sprawdzanie zgodności zachowań funkcjonalnych i niefunkcjonalnych interfejsów z projektem i specyfikacjami;
- budowanie zaufania do jakości interfejsów;
- wykrywanie defektów (które mogą występować w samych interfejsach lub w modułach/systemach);
- zapobieganie przedostawaniu się defektów na wyższe poziomy testowania.

Podobnie jak w przypadku testowania modułowego, istnieją sytuacje w których automatyczne integracyjne testy regresji pozwalają uzyskać pewność, że wprowadzone zmiany nie spowodowały nieprawidłowej pracy dotychczas działających interfejsów, modułów lub systemów.

W niniejszym sylabusie opisano dwa różne poziomy testowania integracyjnego, które mogą być wykonywane w odniesieniu do przedmiotów testów różnej wielkości:

• **Testowanie integracji modułów** skupia się na interakcjach i interfejsach między integrowanymi modułami. Testy tego typu wykonuje się po zakończeniu testowania modułowego (zwykle są to testy automatyczne). W przypadku iteracyjnego i przyrostowego modelu wytwarzania oprogramowania testy integracji modułów są zwykle elementem procesu ciągłej integracji.

• **Testowanie integracji systemów** skupia się na interakcjach i interfejsach między systemami, pakietami i mikrousługami. Testy tego typu mogą również obejmować interakcje z interfejsami dostarczanymi przez organizacje zewnętrzne (np. dostawców usług internetowych — ang. *Web Services*). W takim przypadku organizacja wytwarzająca oprogramowanie nie kontroluje interfejsów zewnętrznych, co może stwarzać wiele problemów wobszarze testowania (związanych np. z usuwaniem defektów w kodzie tworzonym przez organizację zewnętrzną,

które blokują testowanie, bądź zprzygotowywaniem środowisk testowych). Testowanie integracji systemów może odbywać się po zakończeniu testowania systemowego lub równolegle do trwającego testowania systemowego (dotyczy to zarówno sekwencyjnego, jak i przyrostowego wytwarzania oprogramowania).

Podstawa testów

Przykładowe produkty pracy, które mogą być wykorzystywane jako podstawa testów w ramach testowania integracyjnego, to między innymi:

- projekt oprogramowania i systemu;
- diagramy sekwencji;
- specyfikacje interfejsów;
- przypadki użycia;
- architektura na poziomie modułów i systemu;
- przepływy pracy;
- definicje interfejsów zewnętrznych.

Przedmioty testów

Do typowych przedmiotów testów zalicza się:

- podsystemy;
- bazy danych;
- infrastrukturę;
- interfejsy;
- interfejsy programowania aplikacji (ang. Application Programming Interface — API);
- mikrousługi.

Typowe defekty i awarie

Przykładami typowych defektów i awarii wykrywanych w ramach testowania integracji modułów są:

- niepoprawne lub brakujące dane bądź niepoprawne kodowanie danych;
- niepoprawne uszeregowanie lub niepoprawna synchronizacja wywołań interfejsów;
- niezgodności interfejsów;
- błędy komunikacji między modułami;
- brak obsługi lub nieprawidłowa obsługa błędów komunikacji między modułami;

- niepoprawne założenia dotyczące znaczenia, jednostek lub granic danych przesyłanych między modułami.

Przykładami typowych defektów i awarii wykrywanych w ramach testowania integracji systemów są:

- niespójne struktury komunikatów przesyłanych między systemami;
- niepoprawne lub brakujące dane bądź niepoprawne kodowanie danych;
- niezgodność interfejsów;
- błędy komunikacji między systemami;
- brak obsługi lub nieprawidłowa obsługa błędów komunikacji między systemami;
- niepoprawne założenia dotyczące znaczenia, jednostek lub granic danych przesyłanych między systemami;
- nieprzestrzeganie rygorystycznych, obowiązujących przepisów dotyczących zabezpieczeń.

Konkretne podejścia i odpowiedzialności

Testowanie integracji modułów i testowanie integracji systemów powinno koncentrować się na samej integracji. Przykładem takiego podejścia jest integrowanie modułu A z modułem B, podczas którego testy powinny skupiać się na komunikacji między tymi modułami, a nie na funkcjonalności każdego z nich (funkcjonalność ta powinna być przedmiotem testowania modułowego). Analogicznie w przypadku integrowania systemu X z systemem Y testerzy powinni skupiać się na komunikacji między tymi systemami, a nie na funkcjonalności każdego z nich (funkcjonalność ta powinna być przedmiotem testowania systemowego). Na tym poziomie można przeprowadzać testy funkcjonalne, niefunkcjonalne i strukturalne.

Testowanie integracji modułów jest często obowiązkiem programistów, natomiast za testowanie integracji systemów zwykle odpowiadają testerzy. O ile jest to możliwe, testerzy wykonujący testowanie integracji systemów powinni znać architekturę systemu, a ich spostrzeżenia powinny być brane pod uwagę na etapie planowania integracji.

Zaplanowanie testów integracyjnych i strategii integracji przed zbudowaniem modułów lub systemów umożliwia wytworzenie tych modułów lub systemów w sposób zapewniający maksymalną efektywność testowania. Usystematyzowane strategie integracji mogą być oparte na architekturze systemu (np. strategie zstępujące lub wstępujące), zadaniach funkcjonalnych, sekwencjach przetwarzania transakcji bądź innych aspektach systemu lub modułów. Aby uprościć lokalizowanie defektów i umożliwić ich wczesne wykrywanie, integrację należy przeprowadzać metodą przyrostową (np. niewielka liczba jednocześnie dodawanych komponentów lub systemów). Nie zaleca się integracji metodą „wielkiego wybuchu”, polegającą na zintegrowaniu wszystkich modułów lub systemów naraz. W odpowiednim ukierunkowaniu

testowania integracyjnego może również pomóc analiza ryzyka związanego z najbardziej złożonymi interfejsami.

Im szerszy jest zakres integracji, tym trudniej jest wskazać konkretny moduł lub system, w którym wystąpiły defekty, co może prowadzić do wzrostu ryzyka iwydłużenia czasu diagnozowania problemów. Jest to jeden z powodów, dla których powszechnie stosuje się metodę ciągłej integracji, polegającą na integrowaniu oprogramowania moduł po module (np. integracja funkcjonalna). Elementem ciągłej integracji jest często automatyczne testowanie regresji, które w miarę możliwości powinno odbywać się na wielu poziomach testów.

2.2.3. Testowanie systemowe

Cele testowania systemowego

Testowanie systemowe skupia się na zachowaniu i możliwościach całego systemu lub produktu, często zuwzględnieniem całokształtu zadań, jakie może on wykonywać oraz zachowań niefunkcjonalnych, jakie można stwierdzić podczas wykonywania tych zadań. Cele testowania systemowego to między innymi:

- zmniejszanie ryzyka;
- sprawdzanie zgodności zachowań funkcjonalnych i niefunkcjonalnych systemu z projektem i specyfikacjami;
- sprawdzanie kompletności systemu i prawidłowości jego działania;
- budowanie zaufania do jakości systemu jako całości;
- wykrywanie defektów;
- zapobieganie przedostawaniu się defektów na poziom testowania akceptacyjnego lub na produkcję.

W przypadku niektórych systemów celem testowania może być również zweryfikowanie jakości danych. Podobnie jak wprzypadku testowania modułowego i testowania integracyjnego, automatyczne, systemowe testy regresji pozwalają uzyskać pewność, że wprowadzone zmiany nie spowodowały nieprawidłowego działania dotychczasowych funkcji lub całościowej funkcjonalności. Często spotykanym rezultatem testowania systemowego są dane i informacje, na podstawie których interesariusze podejmują decyzje oprzekazaniu systemu do użycia produkcyjnego. Ponadto testowanie systemowe może być niezbędne do spełnienia wymagań wynikających z obowiązujących przepisów prawa lub norm/standardów.

Środowisko testowe powinno w miarę możliwości odzwierciedlać specyfikę środowiska docelowego lub produkcyjnego.

Podstawa testów

Przykładowe produkty pracy, które mogą być wykorzystywane jako podstawa testów w ramach testowania systemowego, to między innymi:

- specyfikacje wymagań (funkcjonalnych i niefunkcjonalnych) dotyczących systemu i oprogramowania;
- raporty z analizy ryzyka;
- przypadki użycia;
- opowieści i historyjki użytkownika;
- modele zachowania systemu;
- diagramy stanów;
- instrukcje obsługi systemu i podręczniki użytkownika.

Przedmioty testów

Do typowych przedmiotów testów dla testów systemowych zalicza się:

- aplikacje;
- systemy łączące sprzęt i oprogramowanie;
- systemy operacyjne;
- system podlegający testowaniu;
- konfiguracja i dane konfiguracyjne systemu.

Typowe defekty i awarie

Przykładami typowych defektów i awarii wykrywanych w ramach testowania systemowego są:

- niepoprawne obliczenia;
- niepoprawne lub nieoczekiwane zachowania funkcjonalne lub niefunkcjonalne systemu;
- niepoprawne przepływy sterowania i/lub przepływy danych w systemie;
- problemy z prawidłowym i kompletnym wykonywaniem całościowych zadań funkcjonalnych;
- problemy z prawidłowym działaniem systemu w środowisku produkcyjnym;
- niezgodność działania systemu z opisami zawartymi w instrukcji obsługi systemu i podręcznikach użytkownika.

Konkretne podejścia i odpowiedzialności

Testowanie systemowe powinno koncentrować się na kompleksowym zbadaniu zachowania systemu jako całości na poziomie zarówno funkcjonalnym, jak i niefunkcjonalnym. W ramach testowania systemowego należy stosować techniki (patrz rozdział 4.), które są najbardziej odpowiednie dla danego aspektu (danych aspektów) systemu będącego przedmiotem testów. W celu sprawdzenia, czy zachowanie funkcjonalne jest zgodne z opisem zawartym w regułach

biznesowych, można wykorzystać np. tablicę decyzyjną.

Testowanie systemowe wykonują zwykle niezależni testerzy. Defekty w specyfikacjach (np. brakujące historyjki użytkownika lub niepoprawnie zdefiniowane wymagania biznesowe) mogą doprowadzić do nieporozumień lub sporów dotyczących oczekiwanego zachowania systemu. Wyniki testów mogą więc zostać zaklasyfikowane jako rezultaty fałszywie pozytywne lub fałszywie negatywne, a sugerowanie się nimi może spowodować odpowiednio stratę czasu lub spadek skuteczności wykrywania defektów.

Między innymi dlatego (aby zmniejszyć częstość występowania powyższych sytuacji) należy zaangażować testerów wprzeglądy, doprecyzowywanie historyjek użytkownika i inne czynności testowe o charakterze statycznym na jak najwcześniejszym etapie prac.

2.2.4. Testowanie akceptacyjne

Cele testowania akceptacyjnego

Testowanie akceptacyjne — podobnie jak testowanie systemowe — skupia się zwykle na zachowaniu i możliwościach całego systemu lub produktu. Cele testowania akceptacyjnego to najczęściej:

- budowanie zaufania do systemu;
- sprawdzanie kompletności systemu i jego prawidłowego działania;
- sprawdzanie zgodności zachowania funkcjonalnego i niefunkcjonalnego systemu ze specyfikacją.

W wyniku testowania akceptacyjnego mogą powstawać informacje pozwalające ocenić gotowość systemu do wdrożenia i użytkowania przez klienta (użytkownika). Podczas testowania akceptacyjnego mogą też zostać wykryte defekty, ale ich wykrywanie często nie jest celem tego poziomu testów (w niektórych przypadkach znalezienie dużej liczby defektów na etapie testowania akceptacyjnego może być nawet uznawane za istotny czynnik ryzyka projektowego). Ponadto testowanie akceptacyjne może być niezbędne do spełnienia wymagań wynikających z obowiązujących przepisów prawa lub norm/standardów.

Najczęściej występujące formy testowania akceptacyjnego to:

- testowanie akceptacyjne przez użytkownika;
- produkcyjne testy akceptacyjne;
- testowanie akceptacyjne zgodności z umową i zgodności z przepisami prawa;
- testowanie alfa i beta.

Formy testowania akceptacyjnego opisano w kolejnych czterech podpunktach.

Testowanie akceptacyjne przez użytkownika (ang. *User Acceptance Testing* – UAT)

Testowanie akceptacyjne systemu przez użytkownika odbywa się w (symulowanym) środowisku produkcyjnym i skupia się zwykle na sprawdzeniu, czy system nadaje się do użytkowania przez przyszłych odbiorców. Głównym celem jest tu budowanie pewności, że system umożliwi użytkownikom spełnienie ich potrzeb, postawionych przed nim wymagań i wykona proces biznesowy z minimalną liczbą problemów, minimalnymi kosztem i ryzykiem.

Produkcyjne testy akceptacyjne (ang. *Operational Acceptance Testing* – OAT)

Testowanie akceptacyjne systemu przez operatorów lub administratorów odbywa się zwykle w (symulowanym) środowisku produkcyjnym. Testy skupiają się na aspektach operacyjnych i mogą obejmować:

- testowanie mechanizmów tworzenia kopii zapasowych i odtwarzania danych;
- instalowanie, odinstalowywanie i aktualizowanie oprogramowania;
- usuwanie skutków awarii;
- zarządzanie użytkownikami;
- wykonywanie czynności pielęgnacyjnych;
- wykonywanie czynności związanych z ładowaniem i migracją danych;
- sprawdzanie, czy występują podatności zabezpieczeń;
- wykonywanie testów wydajnościowych.

Głównym celem produkcyjnych testów akceptacyjnych jest uzyskanie pewności, że operatorzy lub administratorzy systemu będą wstanie zapewnić użytkownikom prawidłową pracę systemu w środowisku produkcyjnym, nawet w wyjątkowych i trudnych warunkach.

Testowanie akceptacyjne zgodności z umową i zgodności z przepisami prawa

Testowanie akceptacyjne zgodności z umową odbywa się zgodnie z kryteriami akceptacji zapisanymi w umowie dotyczącej wytworzenia oprogramowania na zlecenie. Powyższe kryteria akceptacji powinny zostać określone wmomencie uzgadniania przez strony treści umowy. Tego rodzaju testowanie akceptacyjne wykonują często użytkownicy lub niezależni testerzy.

Testowanie akceptacyjne zgodności z przepisami prawa jest wykonywane w kontekście obowiązujących aktów prawnych, takich jak: ustawy, rozporządzenia czy normy bezpieczeństwa. Tego rodzaju testowanie akceptacyjne często wykonują użytkownicy lub niezależni testerzy, a rezultaty mogą być obserwowane lub kontrolowane przez organy nadzoru.

Głównym celem testowania akceptacyjnego zgodności z umową i zgodności z przepisami prawa jest uzyskanie pewności, że osiągnięto zgodność z wymaganiami wynikającymi z obowiązujących umów lub regulacji.

Testowanie alfa i beta

Twórcy oprogramowania przeznaczonego do powszechnej sprzedaży (COTS) często chcą uzyskać informacje zwrotne od potencjalnych lub obecnych klientów, zanim oprogramowanie trafi na rynek. Służą do tego testy alfa i beta.

Testowanie alfa jest wykonywane w siedzibie organizacji wytwarzającej oprogramowanie, ale zamiast zespołu wytwórczego testy wykonują potencjalni lub obecni klienci, i/lub operatorzy bądź niezależni testerzy. Z kolei testowanie beta wykonują obecni lub potencjalni klienci we własnych lokalizacjach. Testy beta mogą być, ale nie muszą, poprzedzone testami alfa.

Jednym z celów testów alfa i beta są budowanie zaufania potencjalnych i aktualnych klientów i/lub operatorów w kwestii korzystania z systemu w normalnych warunkach, w środowisku produkcyjnym, aby osiągnąć swoje cele wkładając w to minimalny wysiłek, koszt i ryzyko. Innym celem tych testów może być wykrywanie błędów związanych z warunkami i środowiskiem (środowiskami), w których system będzie używany, szczególnie wtedy, gdy takie warunki są trudne do odtworzenia dla zespołu projektowego.

Podstawa testów

Przykładowe produkty pracy, które mogą być wykorzystywane jako podstawa testów w ramach dowolnego typu testowania akceptacyjnego, to między innymi:

- procesy biznesowe;
- wymagania użytkowników lub wymagania biznesowe;
- przepisy, umowy, normy i standardy;
- przypadki użycia;
- wymagania systemowe;
- dokumentacja systemu lub podręczniki dla użytkowników;
- procedury instalacji;
- raporty z analizy ryzyka.

Ponadto podstawę testów, z której wyprowadzane są przypadki testowe na potrzeby produkcyjnych testów akceptacyjnych, mogą stanowić następujące produkty pracy:

- procedury tworzenia kopii zapasowych i odtwarzania danych;
- procedury usuwania skutków awarii;
- wymagania niefunkcjonalne;
- dokumentacja operacyjna;
- instrukcje wdrażania i instalacji;

- założenia wydajnościowe;
- pakiety bazodanowe;
- normy, standardy lub przepisy w dziedzinie zabezpieczeń.

Typowe przedmioty testów

Do typowych przedmiotów testów dowolnego typu testów akceptacyjnych zalicza się:

- system podlegający testowaniu;
- konfiguracja i dane konfiguracyjne systemu;
- procesy biznesowe wykonywane na całkowicie zintegrowanym systemie;
- systemy rezerwowe i ośrodki zastępcze (ang. hot site) służące do testowania mechanizmów zapewnienia ciągłości biznesowej i usuwania skutków awarii;
- procesy związane z użyciem produkcyjnym i utrzymaniem;
- formularze;
- raporty;
- istniejące i skonwertowane dane produkcyjne.

Typowe defekty i awarie

Przykładami typowych defektów wykrywanych w ramach różnych typów testowania akceptacyjnego są:

- systemowe przepływy pracy niezgodne zwymaganiami biznesowymi lub wymaganiami użytkowników;
- niepoprawnie zaimplementowane reguły biznesowe;
- niespełnienie przez system wymagań umownych lub prawnych;
- awarie niefunkcjonalne, takie jak podatności zabezpieczeń, niedostateczna wydajność pod dużym obciążeniem bądź nieprawidłowe działanie na obsługiwanej platformie.

Konkretne podejścia i obowiązki

Testowanie akceptacyjne często spoczywa na klientach, użytkownikach biznesowych, właścicielach produktów lub operatorach systemów, ale wproces ten mogą być również zaangażowani inni interesariusze. Testowanie akceptacyjne zazwyczaj uznaje się za ostatni poziom sekwencyjnego cyklu życia oprogramowania, ale może ono również odbywać się na innych jego etapach, np.:

- testowanie akceptacyjne oprogramowania „do powszechnej sprzedaży” może odbywać się podczas jego instalowania lub integrowania;

• testowanie akceptacyjne nowego udoskonalenia funkcjonalnego może mieć miejsce przed rozpoczęciem testowania systemowego.

W iteracyjnych modelach wytwarzania oprogramowania zespoły projektowe mogą stosować na zakończenie każdej iteracji różne formy testowania akceptacyjnego, takie jak testy skupiające się na weryfikacji zgodności nowej funkcjonalności zkryteriami akceptacji bądź testy skupiające się na walidacji nowej funkcjonalności z punktu widzenia potrzeb użytkowników. Ponadto na końcu każdej iteracji, po ukończeniu każdej iteracji lub po wykonaniu serii iteracji, mogą być wykonywane testy alfa i beta, a także testy akceptacyjne wykonywane przez użytkownika, produkcyjne testy akceptacyjne oraz testy akceptacyjne zgodności z umową i zgodności z przepisami prawa.

2.3 Typ testów

Typ testów to grupa dynamicznych czynności testowych wykonywanych z myślą o przetestowaniu określonych charakterystyk systemu/oprogramowania (lub jego części) zgodnie z określonymi celami testów. Celem testów może być między innymi:

- ocena funkcjonalnych charakterystyk jakościowych takich jak: kompletność, prawidłowość i adekwatność;
- dokonanie oceny niefunkcjonalnych charakterystyk jakościowych, w tym parametrów takich jak: niezawodność, wydajność, bezpieczeństwo, kompatybilność czy też użyteczność;
- ustalenie, czy struktura lub architektura komponentu lub systemu jest poprawna, kompletna i zgodna ze specyfikacjami;
- dokonanie oceny skutków zmian, np. potwierdzenie usunięcia defektów (testowanie potwierdzające) lub wyszukanie niezamierzonych zmian w sposobie działania wynikających z modyfikacji oprogramowania lub środowisku (testowanie regresji).

2.3.1. Testowanie funkcjonalne

Testowanie funkcjonalne systemu polega na wykonaniu testów, które umożliwiają dokonanie oceny funkcji, jakie system ten powinien realizować. Wymagania funkcjonalne mogą być opisane w produktach pracy takich jak: specyfikacje wymagań biznesowych, opowieści, historyjki użytkownika, przypadki użycia lub specyfikacje funkcjonalne, ale zdarza się również, że występują one w postaci nieudokumentowanej. Funkcje opisują to, „co” powinien robić dany system.

Testy funkcjonalne należy wykonywać na wszystkich poziomach testów (np. testy dotyczące modułów mogą opierać się na specyfikacjach modułów), jednakże z zastrzeżeniem, że testy wykonywane na poszczególnych poziomach skupiają się na różnych zagadnieniach.

Testowanie funkcjonalne uwzględnia zachowanie oprogramowania, w związku z czym do wyprowadzania warunków testowych i przypadków testowych dotyczących funkcjonalności komponentu lub systemu można używać technik czarnoskrzynkowych. Staranność testowania

funkcjonalnego można zmierzyć na podstawie pokrycia funkcjonalnego. Termin „pokrycie funkcjonalne" oznacza stopień, w jakim został przetestowany określony typ elementu funkcjonalnego, wyrażony jako procent elementów danego typu pokrytych przez testy. Dzięki możliwości śledzenia powiązań między testami a wymaganiami funkcjonalnymi można na przykład obliczyć, jaki procent wymagań został uwzględniony w ramach testowania, a w rezultacie zidentyfikować ewentualne luki w pokryciu.

Do projektowania i wykonywania testów funkcjonalnych mogą być potrzebne specjalne umiejętności lub specjalna wiedza taka jak znajomość konkretnego problemu biznesowego rozwiązywanego przy pomocy danego oprogramowana (np. oprogramowania do modelowania geologicznego dla przemysłu naftowego i gazowego) bądź konkretnej roli, jaką spełnia dane oprogramowanie (np. gra komputerowa zapewniająca interaktywną rozrywkę).

2.3.2. Testowanie niefunkcjonalne

Celem testowania niefunkcjonalnego jest dokonanie oceny charakterystyk systemów ioprogramowania, takich jak: użyteczność, wydajność, bezpieczeństwo itd. Klasyfikację charakterystyk jakościowych oprogramowania zawiera standard ISO/IEC25010. Testowanie niefunkcjonalne pozwala sprawdzić to, „jak dobrze" zachowuje się dany system.

Wbrew mylnemu przekonaniu testowanie niefunkcjonalne można i często należy wykonywać na wszystkich poziomach testów. Ponadto powinno ono odbywać się na jak najwcześniejszym etapie, ponieważ zbyt późne wykrycie defektów niefunkcjonalnych może być bardzo dużym zagrożeniem dla powodzenia projektu.

Do wyprowadzania warunków testowych i przypadków testowych na potrzeby testowania niefunkcjonalnego można używać technik czarnoskrzynkowych. Przykładem może być zastosowanie analizy wartości brzegowych do zdefiniowania warunków skrajnych dotyczących testów wydajnościowych.

Staranność testowania niefunkcjonalnego można zmierzyć na podstawie pokrycia niefunkcjonalnego. Termin „pokrycie niefunkcjonalne" oznacza stopień, w jakim został przetestowany określony typ elementu niefunkcjonalnego, wyrażony jako procent elementów danego typu pokrytych przez testy. Dzięki możliwości śledzenia powiązań między testami a typami urządzeń obsługiwanych przez aplikację mobilną można na przykład obliczyć, jaki procent urządzeń został uwzględniony w ramach testowania kompatybilności, a w rezultacie zidentyfikować ewentualne luki w pokryciu.

Do projektowania i wykonywania testów niefunkcjonalnych mogą być potrzebne specjalne umiejętności lub specjalna wiedza — taka jak znajomość słabych punktów charakterystycznych dla danego projektu lub danej technologii (np. podatności zabezpieczeń związanych zokreślonymi językami programowania) bądź konkretnej grupy użytkowników (np. profili użytkowników systemów do zarządzania placówkami opieki zdrowotnej).

2.3.3. Testowanie białoskrzynkowe

W przypadku testowania białoskrzynkowego warunki testowe wyprowadza się na podstawie struktury wewnętrznej lub implementacji danego systemu. Struktura wewnętrzna może obejmować kod, architekturę, przepływy pracy i/lub przepływy danych w obrębie systemu.

Staranność testowania białoskrzynkowego można zmierzyć na podstawie pokrycia strukturalnego. Termin „pokrycie strukturalne" oznacza stopień, w jakim został przetestowany określony typ elementu strukturalnego, wyrażony jako procent elementów danego typu pokrytych przez testy.

Na poziomie testowania modułów pokrycie kodu określa się na podstawie procentowej części kodu danego modułu, która została przetestowana. Wartość tę można mierzyć w kategoriach różnych aspektów kodu (przedmiotów pokrycia), takich jak procent instrukcji wykonywalnych lub procent wyników decyzji przetestowanych w danym module. Powyższe rodzaje pokrycia nazywa się zbiorczo pokryciem kodu. Na poziomie testowania integracji modułów, testowanie białoskrzynkowe może odbywać się na podstawie architektury systemu (np. interfejsów między modułami), a pokrycie strukturalne może być mierzone w kategoriach procentowego udziału przetestowanych interfejsów.

Do projektowania i wykonywania testów białoskrzynkowych mogą być potrzebne specjalne umiejętności lub specjalna wiedza, np.: znajomość struktury kodu (umożliwiająca np. użycie narzędzi do pomiaru pokrycia kodu), wiedza o sposobie przechowywania danych (pozwalająca np. ocenić zapytania do baz danych) bądź sposób korzystania z narzędzi do pomiaru pokrycia i poprawnego interpretowania generowanych przez nie rezultatów.

2.3.4. Testowanie związane ze zmianami

Po wprowadzeniu wsystemie zmian mających na celu usunięcie defektów bądź dodanie lub zmodyfikowanie funkcjonalności należy przeprowadzić testy, które potwierdzą, że wprowadzone zmiany faktycznie skutkowały naprawieniem defektu lub poprawną implementacją odpowiedniej funkcjonalności, a przy tym nie wywołały żadnych nieprzewidzianych, niekorzystnych zachowań oprogramowania.

Testowanie potwierdzające. Po naprawieniu defektu można przetestować oprogramowanie przy użyciu wszystkich przypadków testowych, które wcześniej nie zostały zaliczone z powodu wystąpienia tego defektu, a powinny zostać ponownie wykonane w nowej wersji oprogramowania. Oprogramowanie może być również testowane nowymi testami, jeśli na przykład defektem była brakująca funkcjonalność. Absolutnym minimum jest ponowne wykonanie w nowej wersji oprogramowania kroków niezbędnych do odtworzenia awarii wywoływanej przez dany defekt. Celem testu potwierdzającego jest sprawdzenie, czy pierwotny defekt został pomyślnie usunięty.

Testowanie regresji. Istnieje ryzyko, że zmiana (poprawka lub inna modyfikacja) wprowadzona w jednej części kodu wpłynie przypadkowo na zachowanie innych części kodu w tym samym module, w innych modułach tego samego systemu, a nawet w innych systemach. Ponadto należy

wziąć pod uwagę zmiany dotyczące środowiska, takie jak wprowadzenie nowej wersji systemu operacyjnego lub systemu zarządzania bazami danych. Powyższe, niezamierzone skutki uboczne, są nazywane regresjami, atestowanie regresji polega na ponownym wykonaniu testów w celu ich wykrycia.

Testowanie potwierdzające i testowanie regresji można wykonywać na wszystkich poziomach testów.

Zwłaszcza w iteracyjnych i przyrostowych modelach wytwarzania oprogramowania (np. w modelu zwinnym) nowe funkcjonalności, zmiany dotychczasowych funkcjonalności oraz refaktoryzacja powodują częste zmiany kodu, które pociągają za sobą konieczność przeprowadzenia odpowiednich testów. Z uwagi na ciągłą ewolucję systemu, testowanie potwierdzające i testowanie regresji są bardzo istotne, szczególnie w przypadku systemów związanych z Internetem rzeczy (IoT), w których poszczególne obiekty (np. urządzenia) są często aktualizowane lub wymieniane.

Zestawy testów regresji są wykonywane wielokrotnie i zwykle ewoluują dość wolno, w związku z czym testowanie regresji świetnie nadaje się do automatyzacji – dlatego też automatyzację tego rodzaju testów należy rozpocząć w początkowym etapie projektu.

2.3.5. Poziomy testów a typy testów

Każdy z wymienionych powyżej typów testów można wykonywać na dowolnym poziomie testów. Aby zilustrować tę prawidłowość, poniżej przedstawiono przykłady testów funkcjonalnych, niefunkcjonalnych i białoskrzynkowych oraz testów związanych ze zmianami, które są wykonywane na wszystkich poziomach testów w odniesieniu do aplikacji bankowej. Poniżej podano przykłady testów funkcjonalnych wykonywanych na różnych poziomach testów:

- na potrzeby testowania modułowego projektowane są testy odzwierciedlające sposób, w jaki dany moduł powinien obliczać odsetki składane;
- na potrzeby testowania integracji modułów projektowane są testy odzwierciedlające sposób, w jaki informacje na temat konta pozyskane poprzez interfejs użytkownika są przekazywane do warstwy logiki biznesowej;
- na potrzeby testowania systemowego projektowane są testy odzwierciedlające sposób, w jaki posiadacze rachunków mogą składać wnioski o przyznanie linii kredytowej na rachunku bieżącym;
- na potrzeby testowania integracji systemów projektowane są testy odzwierciedlające sposób, wjaki dany system sprawdza zdolność kredytową posiadacza rachunku przy użyciu zewnętrznej mikrousługi;
- na potrzeby testowania akceptacyjnego projektowane są testy odzwierciedlające sposób, w jaki pracownik banku zatwierdza lub odrzuca wniosek kredytowy.

Poniżej opisano przykłady testów niefunkcjonalnych przeprowadzanych na różnych poziomach testów:

• na potrzeby testowania modułowego projektowane są testy wydajnościowe, które umożliwiają ocenę liczbę cykli procesora niezbędnych do wykonania złożonych obliczeń dotyczących łącznej kwoty odsetek;

• na potrzeby testowania integracji modułów projektowane są testy zabezpieczeń, które mają na celu wykrycie podatności zabezpieczeń związanych z przepełnieniem bufora danymi przekazywanymi z interfejsu użytkownika do warstwy logiki biznesowej;

• na potrzeby testowania systemowego projektowane są testy przenaszalności, które umożliwiają sprawdzenie, czy warstwa prezentacji działa we wszystkich obsługiwanych przeglądarkach i urządzeniach przenośnych;

• na potrzeby testowania integracji systemów projektowane są testy niezawodności, które pozwalają na dokonanie oceny odporności systemu w przypadku braku odpowiedzi ze strony mikrousługi służącej do sprawdzania zdolności kredytowej;

• na potrzeby testowania akceptacyjnego projektowane są testy użyteczności, które pozwalają ocenić ułatwienia dostępu dla osób niepełnosprawnych zastosowane w interfejsie do przetwarzania kredytów po stronie banku.

• na potrzeby testowania modułowego projektowane są testy, których celem jest zapewnienie pełnego pokrycia instrukcji kodu i decyzji we wszystkich modułach wykonujących obliczenia finansowe;

• na potrzeby testowania integracji modułów projektowane są testy, które umożliwiają sprawdzenie, w jaki sposób każdy ekran interfejsu wyświetlanego w przeglądarce przekazuje dane prezentowane na następnym ekranie oraz do warstwy logiki biznesowej;

• na potrzeby testowania systemowego projektowane są testy, których celem jest zapewnienie pokrycia możliwych sekwencji stron internetowych wyświetlanych podczas składania wniosku o przyznanie linii kredytowej;

• na potrzeby testowania integracji systemów projektowane są testy, których celem jest sprawdzenie wszystkich możliwych typów zapytań wysyłanych do mikrousługi służącej do sprawdzania zdolności kredytowej;

• na potrzeby testowania akceptacyjnego projektowane są testy, których celem jest pokrycie wszystkich obsługiwanych struktur i zakresów wartości dla plików z danymi finansowymi używanych w przelewach międzybankowych.

Ostatnia grupa zawiera przykłady testów związanych ze zmianami przeprowadzanych na różnych poziomach testów:

• na potrzeby testowania modułowego projektowane są automatyczne testy regresji dotyczące poszczególnych modułów (testy te zostaną następnie uwzględnione wstrukturze ciągłej integracji);

• na potrzeby testowania integracji modułów projektowane są testy służące do potwierdzania skuteczności wprowadzonych poprawek związanych z defektami w interfejsach, w miarę umieszczania takich poprawek w repozytorium kodu;

• na potrzeby testowania systemowego ponownie wykonywane są wszystkie testy dotyczące danego przepływu pracy, jeśli dane czy sposób ich prezentacji na którymkolwiek z ekranów objętych tym przepływem pracy uległ zmianie;

• na potrzeby testowania integracji systemów codziennie wykonywane są ponownie testy aplikacji współpracującej z mikrousługą do sprawdzania zdolności kredytowej (w ramach ciągłego wdrażania tej mikrousługi);

• na potrzeby testowania akceptacyjnego wszystkie wcześniej niezaliczone testy są wykonywane ponownie (po usunięciu defektu wykrytego w ramach testowania akceptacyjnego).

Powyżej przedstawiono przykłady wszystkich typów testów wykonywanych na wszystkich poziomach testów, jednak nie każde oprogramowanie wymaga uwzględnienia każdego typu testów na każdym poziomie. Ważne jest to, aby na poszczególnych poziomach zostały wykonane odpowiednie testy — dotyczy to zwłaszcza pierwszego z poziomów testów, na jakim występuje dany typ testów.

2.4 Testowanie pielęgnacyjne

Po wdrożeniu w środowisku produkcyjnym oprogramowanie lub system wymaga dalszej pielęgnacji. Różnego rodzaju zmiany — związane np. z usuwaniem defektów wykrytych podczas użytkowania produkcyjnego, dodaniem nowej funkcjonalności bądź usuwaniem lub modyfikowaniem funkcjonalności już istniejącej — są praktycznie nieuniknione. Ponadto pielęgnacja jest niezbędna do utrzymania lub poprawy wymaganych niefunkcjonalnych charakterystyk jakościowych oprogramowania lub systemu przez cały cykl jego życia — zwłaszcza w zakresie takich parametrów jak: wydajność, kompatybilność, niezawodność, bezpieczeństwo i przenaszalność.

Po dokonaniu każdej zmiany w fazie pielęgnacji, należy wykonać testowanie pielęgnacyjne, którego celem jest zarówno sprawdzenie, czy zmiana została wprowadzona pomyślnie, jak i wykrycie ewentualnych, niezamierzonych skutków ubocznych (np. regresji) wniezmienionych częściach systemu (czyli zwykle wwiększości jego obszarów). Testowanie pielęgnacyjne obejmuje więc zarówno te części systemu, które zostały zmienione, jak i części niezmienione, na które zmiany mogły mieć wpływ. Pielęgnacja może być wykonywana zarówno planowo (w związku z nowymi wersjami), jak i w sposób niezaplanowany (w związku z poprawkami doraźnymi — ang. *hotfix*).

Wydanie pielęgnacyjne może wymagać wykonania testów pielęgnacyjnych na wielu poziomach testów i z wykorzystaniem różnych typów testów — zależnie od zakresu wprowadzonych zmian. Na zakres testowania pielęgnacyjnego wpływają między innymi:

- poziom ryzyka związanego ze zmianą (np. stopień, wjakim zmieniony obszar oprogramowania komunikuje się z innymi modułami lub systemami);
- wielkość dotychczasowego systemu;
- wielkość wprowadzonej zmiany.

2.4.1. Zdarzenia wywołujące pielęgnację

Istnieje kilka powodów, dla których wykonuje się pielęgnację oprogramowania, atym samym testowanie pielęgnacyjne. Dotyczy to zarówno zmian planowanych, jak i nieplanowanych.

Zdarzenia wywołujące pielęgnację można podzielić na następujące kategorie:

• Modyfikacja. Ta kategoria obejmuje między innymi: zaplanowane udoskonalenia (np. w postaci nowych wersji oprogramowania), zmiany korekcyjne i awaryjne, zmiany środowiska operacyjnego (np. planowe uaktualnienia systemu operacyjnego lub bazy danych), uaktualnienia oprogramowania do powszechnej sprzedaży (COTS) oraz poprawki usuwające defekty i podatności zabezpieczeń.

• **Migracja.** Ta kategoria obejmuje między innymi przejście z jednej platformy na inną, co może wiązać się zkoniecznością przeprowadzenia testów produkcyjnych nowego środowiska i zmienionego oprogramowania bądź testów konwersji danych (w przypadku migracji danych z innej aplikacji do pielęgnowanego systemu).

• **Wycofanie.** Ta kategoria dotyczy sytuacji, w której aplikacja jest wycofywana z użytku. Kiedy aplikacja lub system jest wycofywany, może to wymagać testowania migracji lub archiwizacji danych, jeśli zachodzi potrzeba ich przechowywania przez dłuższy czas. –

- Testowanie procedur odzyskiwania/pozyskiwania zarchiwizowanych danych przez dłuższy czas może także być konieczne.

- Dodatkowo nieodzowne może być uwzględnienie testów regresji, aby zapewnić dalszą prawidłową pracę funkcjonalności pozostających w użyciu.

W przypadku systemów związanych z Internetem rzeczy (IoT – *Internet of Things*) testowanie pielęgnacyjne może być konieczne po wprowadzeniu wsystemie zupełnie nowych lub zmodyfikowanych elementów, takich jak urządzenia sprzętowe czy usługi programowe. Podczas testowania pielęgnacyjnego tego typu systemów szczególny nacisk kładzie się na testowanie integracyjne na różnych poziomach (np. na poziomie sieci i aplikacji) oraz na aspekty związane z zabezpieczeniami, szczególnie w zakresie danych osobowych.

2.4.2. Analiza wpływu związana z pielęgnacją

Analiza wpływu pozwala ocenić zmiany wprowadzone w wersji pielęgnacyjnej pod kątem zarówno skutków zamierzonych, jak i spodziewanych lub potencjalnych skutków ubocznych, a

także umożliwia zidentyfikowanie obszarów systemu, na które będą miały wpływ wprowadzone zmiany. Ponadto może pomóc w zidentyfikowaniu wpływu zmiany na dotychczasowe testy. Skutki uboczne zmiany oraz obszary systemu, na które może ona wpływać, należy przetestować pod kątem regresji, przy czym czynność ta może być poprzedzona aktualizacją istniejących testów, na które oddziałuje dana zmiana.

Analizę wpływu można przeprowadzić przed dokonaniem zmiany, aby ustalić, czy zmianę tę należy faktycznie wprowadzić (z uwagi na potencjalne konsekwencje dla innych obszarów systemu).

Przeprowadzenie analizy wpływu może być utrudnione, jeśli:

- specyfikacje (np. wymagania biznesowe, historyjki użytkownika, architektura) są nieaktualne lub niedostępne;
- przypadki testowe nie zostały udokumentowane lub są nieaktualne;
- nie stworzono możliwości dwukierunkowego śledzenia powiązań między testami a podstawą testów;
- wsparcie narzędziowe nie istnieje lub jest niewystarczające;
- zaangażowane osoby nie dysponują wiedzą z danej dziedziny i/lub na temat danego systemu;
- podczas wytwarzania oprogramowania poświęcono zbyt mało uwagi jego charakterystyce jakościowej w zakresie utrzymywalności.