Certyfikowany tester

Plan poziomu podstawowego

Wersja 2011.1.3

Historia zmian

Wersja	Data	Uwagi
0.1	01.01.2010 – 15.06.2010	Tłumaczenie autorskie: Jan Sabak, wersja 2007
0.2	15.06.2010 – 15.12.2010	Udostępnienie wersji 0.1 na stronie internetowej, zbieranie uwag
0.4	15.12.2010 – 15.02.2011	Tomasz Watras – przegląd
0.5	15.02.2011 – 15.05. 2011	Lucjan Stapp – przegląd
0.6	15.10.2011	Jan Sabak - uaktualnienie do wersji 2011
0.7	22.03.2012	Lucjan Stapp - scalenie dokumentu
0.8	17.05.2012	Lucjan Stapp – rozdziały 7 -12, stworzenie indeksu
0.9	1.06.2012- 10.06.2012	Jan Sabak, Lucjan Stapp – ostateczny przegląd dokumentu
1.0	15.06.2012	Wersja dostarczona do SJSI
1.0	3.07.2012	Zatwierdzenie przez Zarząd SJSI
1.0	20.08.2012 – 13.09.2012	Radosław Smilgin – przegląd
1.1	25.09.2012	Lucjan Stapp - zatwierdzenie zmian i poprawek
1.2	13.10.2017	Jan Sabak – drobne poprawki merytoryczne
1.3	11.11.2017	Jan Sabak – drobne poprawki edycyjne

Spis treści

Podziękowania

Podziękowania dla

Grupy Roboczej ISTQB dla poziomu podstawowego (wersja 2011): Thomas Müller (przewodniczący), Debra Friedenberg

przy współpracy następujących osób: Armin Beer, Rex Black, Julie Gardiner, Judy McCay, Tuula Paakkonen, Eric Rio udu Cosquier, Hans Schaefer, Stephanie Ulrich, Erik van Veendendal.

Grupa Robocza wyraża podziękowania przedstawicielom Rad Krajowych za sugestie przy tworzeniu aktualnej wersji sylabusa.

Tłumaczenie na język polski zostało wykonane przez Jana Sabaka. Przeglądów dokonali Tomasz Watras, Lucjan Stapp i Radosław Smilgin.

Wstęp do planu

Cel dokumentu

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

Podstawowy Poziom Certyfikowanego Testera w Testowaniu Oprogramowania

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

Cele uczenia się/poziom wiedzy

Dla każdej sekcji planu podane są następujące poziomy poznawcze:
K1: zapamiętać, rozpoznać, wiedzieć;
K2: podsumowywać, klasyfikować, porównywać, przypisywać, przeciwstawiać, podawać przykłady, interpretować, reprezentować, wnioskować, kategoryzować;
K3: zastosować;
K4: analizować.
Dalsze szczegóły i przykłady celów uczenia się podano w Załączniku B. Wszystkie pojęcia wymienione w „Terminologii", zaraz poniżej nagłówków rozdziałów, powinny być zapamiętane, nawet, jeśli nie wspomniano o tym wyraźnie, w celach uczenia się.

Egzamin

Egzamin na Certyfikat Podstawowy będzie oparty na niniejszym konspekcie. Odpowiedzi na pytania egzaminacyjne mogą wymagać skorzystania z materiału bazującego na więcej niż jednym rozdziale niniejszego planu. W egzaminie uwzględnione są wszystkie rozdziały planu.
Format egzaminu to test wielokrotnego wyboru.
Egzaminy można zdawać podczas akredytowanego kursu szkoleniowego lub przystępować do nich niezależnie (np. w centrum egzaminacyjnym).

Akredytacja

Dostawcy szkoleń, których materiał kursowy odpowiada planowi, mogą być akredytowani przez narodową organizację uznawaną przez ISTQB. Wytyczne odnośnie akredytacji powinno się uzyskać od organizacji lub ciała, które dokonuje akredytacji. Kurs akredytowany

uznawany jest jako odpowiadający temu planowi. Zezwala się, aby zdawać egzamin ISTQB® jako część kursu.
Dalsze wskazówki odnośnie dostawców szkoleń podano w Załączniku D.

Poziom uszczegółowienia

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

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

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

Organizacja planu

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

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

pokazuje, że rozdział 2 ma poziom wiedzy K1 (zakłada się zawieranie niższych, gdy pokazany jest wyższy poziom) i K2 (ale nie K3) i zaplanowano 135 minut na nauczanie materiału w nim zawartego.
Wewnątrz każdego rozdziału jest wiele sekcji. Każda sekcja ma również cele uczenia się i ilość wymaganego czasu. Podsekcje, dla których nie podano czasu, wliczają się do czasu dla sekcji.

1. **Podstawy testowania**

K2 155min

Cele nauczania dla podstaw testowania.

Te cele określają, co będziesz potrafił po zakończeniu modułu.

1.1 Dlaczego testowanie jest niezbędne? (K2)

LO-1.1.1 Kandydat potrafi opisać - podając przykłady - w jaki sposób usterka w oprogramowaniu może wyrządzić szkodę osobie, środowisku lub firmie. (K2)
LO-1.1.2 Kandydat rozróżnia przyczynę usterki od jej skutków. (K2)
LO-1.1.3 Kandydat potrafi podać powody tego, że testowanie jest niezbędne opierając się na przykładach. (K2)
LO-1.1.4 Kandydat potrafi uzasadnić, że testowanie stanowi część zapewnienia jakości i podać przykłady tego, w jaki sposób testowanie przyczynia się do zwiększenia jakości.
LO-1.1.5 Kandydat potrafi wyjaśnić i porównać pojęcia pomyłka, usterka, awaria oraz odpowiadające im pojęcia błąd i pluskwa, podając przykłady. (K2)

1.2 Co to jest testowanie? (K2)

LO-1.2.1 Kandydat pamięta ogólne cele testowania. (K1)
LO-1.2.2 Kandydat potrafi podać przykłady celów testowania w różnych fazach cyklu życia oprogramowania. (K2)
LO-1.2.3 Kandydat rozróżnia testowanie od debagowania. (K2)

1.3 Siedem zasad testowania (K2)

LO-1.3.1 Kandydat potrafi wyjaśnić siedem zasad testowania.

1.4 Podstawowy proces testowy (K1)

LO-1.4.1 Kandydat pamięta pięć podstawowych czynności testowych i odpowiadające im zadania od planowania do zamknięcia testów. (K1)

1.5 Psychologia testowania (K2)

LO-1.5.1 Kandydat pamięta czynniki psychologiczne, od których zależy sukces testowania. (K1)
LO-1.5.2 Kandydat potrafi pokazać różnice w nastawieniu testera i programisty. (K2)

1.1 Dlaczego testowanie jest niezbędne?

K2 20min

Pojęcia
pluskwa, defekt, błąd, awaria, usterka, pomyłka, jakość, ryzyko

K1

1.1.1 Kontekst systemów softwarowych

Systemy softwarowe są integralną częścią naszego życia, od aplikacji biznesowych (np. w bankowości) po produkty użytkowe (np. samochody). Większość ludzi spotkała się z oprogramowaniem, które nie działało tak jak powinno. Oprogramowanie, które działa niepoprawnie może spowodować wiele problemów, stratę pieniędzy, czasu lub utratę reputacji biznesowej. Może nawet spowodować utratę zdrowia lub życia.

K2

1.1.2 Przyczyny usterek w oprogramowaniu

Człowiek może popełnić błąd (pomyłkę), która powoduje powstanie defektu (usterki, pluskwy) w kodzie programu lub dokumencie. Jeżeli kod zawierający defekt zostaje wykonany, system nie zrobi tego co powinien (lub wykona to czego nie powinien) powodując awarię. Usterki w oprogramowaniu, systemach lub dokumentach mogą prowadzić do wystąpienia awarii, ale nie wszystkie usterki powodują awarie.

Defekty istnieją, ponieważ ludzie są omylni, działają pod presją, pracują ze złożonym kodem, w skomplikowanej infrastrukturze, ze zmieniającymi się technologiami oraz z dużą ilością interakcji pomiędzy systemami.

Awarie mogą zostać spowodowane też przez warunki środowiskowe. Na przykład promieniowanie, pole magnetyczne i elektryczne oraz zanieczyszczenia mogą spowodować awarie oprogramowania wbudowanego lub wpływać na oprogramowanie przez zmianę warunków działania sprzętu.

K2

1.1.3 Rola testowania w rozwoju, utrzymaniu i użytkowaniu oprogramowania

Rygorystyczne testy systemów i dokumentacji mogą pomóc w zmniejszeniu ryzyka wystąpienia problemów podczas użytkowania oprogramowania i przyczynić się do podniesienia jakości sytemu, jeżeli znalezione usterki zostaną poprawione przed wdrożeniem systemu do użytkowania.

Testowanie oprogramowania może być wymagane również przez kontrakt lub ze względu na uwarunkowania prawne lub standardy obowiązujące w danej gałęzi przemysłu.

1.1.4 Testowanie a jakość

Za pomocą testów można zmierzyć jakość oprogramowania wyrażoną przez ilość znalezionych usterek zarówno dla funkcjonalnych jak i niefunkcjonalnych wymagań i atrybutów oprogramowania (np. niezawodności, użyteczności, efektywności, pielęgnowalności lub przenaszalności). Więcej informacji na temat testów niefunkcjonalnych znajduje się w rozdziale 2, a na temat atrybutów jakościowych oprogramowania w standardzie "Software Engineering – Software Product Quality" ISO 9126. K2

Testowanie może budować zaufanie do jakości oprogramowania jeżeli osoby testujące znajdują mało usterek lub nie znajdują ich wcale. Poprawnie zaprojektowany test, który jest zaliczony [1] obniża ogólny poziom ryzyka w systemie. Kiedy testowanie wykrywa usterki, jakość systemu podnosi się wraz z ich naprawieniem. Samo testowanie nie podnosi jakości oprogramowania i dokumentacji.

Powinno się wyciągać wnioski z poprzednich projektów. Przez zrozumienie pierwotnych przyczyn usterek można doskonalić procesy, co z kolei powinno przeciwdziałać ponownemu pojawianiu się podobnych usterek i w konsekwencji podnosić jakość przyszłych systemów. Jest to jeden z aspektów zapewnienia jakości.

Testowanie powinno być zintegrowane z innymi działaniami w ramach zapewnienia jakości (np. standardami kodowania, szkoleniami i analizą defektów).

K2

1.1.5 Jak dużo testowania jest potrzebne?

Podczas podejmowania decyzji jak dużo testów należy wykonać, powinno się wziąć pod uwagę poziom ryzyka, włączając w to ryzyko techniczne, ryzyko związane z bezpieczeństwem, a także ryzyko biznesowe oraz ograniczenia projektowe takie jak czas i budżet. (Pojęcie ryzyka omówione jest w rozdziale 5.)

Testowanie powinno dostarczać interesariuszom informacji wystarczających do podjęcia świadomych decyzji o dopuszczeniu testowanego oprogramowania lub systemu do następnej fazy rozwoju lub przekazaniu go klientowi.

1.2 Co to jest testowanie?

K2 30min

Pojęcia
debagowanie, wymaganie, przegląd, przypadek testowy, testowanie, cel testu

Wprowadzenie

Za testowanie najczęściej uważa się tylko wykonywanie testów, to jest uruchamianie oprogramowania. Wykonywanie testów stanowi tylko część testowania, bowiem istnieją jeszcze inne czynności związane z testowaniem.

Czynności testowania występują zarówno przed, jak i po wykonywaniu testów. Należą do nich: planowanie i nadzór, wybór warunków testowych, projektowanie i wykonywanie przypadków testowych, sprawdzanie wyników, ocena spełnienia kryteriów zakończenia, raportowanie procesu testowania i testowanego systemu oraz kończenie i zamykanie testów (po zakończeniu fazy testów). Do testowania zalicza się także przeglądy dokumentacji i kodu źródłowego oraz analizę statyczną.

Testy dynamiczne i statyczne mogą służyć jako środek do osiągnięcia podobnych celów. Oba rodzaje testów dostarczają informacji pozwalających na doskonalenie zarówno testowanego systemu jak i procesów rozwoju i testowania oprogramowania.

Istnieją różne cele testowania:

- znajdowanie usterek
- nabieranie zaufania do poziomu jakości
- dostarczanie informacji potrzebnych do podejmowania decyzji
- zapobieganie defektom

Proces myślowy oraz czynności związane z projektowaniem testów wcześnie w cyklu życia oprogramowania (weryfikacja podstawy testów przez projektowanie testów) mogą zapobiec wprowadzeniu usterek do kodu. Przeglądy dokumentów (np. specyfikacji wymagań) oraz identyfikacja i rozwiązywanie problemów również pomagają w przeciwdziałaniu pojawianiu się defektów w kodzie.

Różne punkty widzenia pozwalają na wzięcie pod uwagę w testach różnych celów. Na przykład w testowaniu wytwórczym (np. testowaniu modułowym, integracyjnym lub systemowym) głównym celem może być wywołanie tylu awarii ile się da, żeby zidentyfikować i poprawić usterki występujące w oprogramowaniu. W testach akceptacyjnych głównym celem może być potwierdzenie, że system działa tak jak powinien oraz nabranie pewności, że spełnia wymagania. W niektórych przypadkach celem testowania może być ocena jakości oprogramowania (bez intencji naprawiania defektów), by dostarczyć interesariuszom informacji o ryzyku związanym z wydaniem systemu w danej chwili. Testowanie pielęgnacyjne często zawiera testy sprawdzające, czy nie wprowadzono nowych usterek podczas wykonywania zmian. Głównym celem testowania produkcyjnego może być ocena atrybutów systemu takich jak niezawodność lub dostępność.

Debagowanie różni się od testowania. Testowanie dynamiczne może pokazać awarie, których źródłem są usterki. Debagowanie jest czynnością programistyczną, która znajduje, analizuje i umożliwia usunięcie przyczyny awarii. Późniejsze testy potwierdzające (retesty) wykonywane przez testera gwarantują, że poprawka rzeczywiście usunęła usterkę. Odpowiedzialność za każdą z tych czynności jest zwykle inna tj. testerzy testują, a programiści debagują.

Proces testowania i czynności z nim związane omówione są w podrozdziale 1.4.

1.3 Ogólne zasady testowania

K2 35min

Pojęcia
testowanie gruntowne

Zasady

W ciągu ostatnich czterdziestu lat powstała pewna ilość zasad testowania, które zawierają ogólne wskazówki przydatne przy każdych testach.

Zasada 1 - Testowania ujawnia usterki

Testowanie może pokazać, że istnieją usterki, ale nie może dowieść, że oprogramowanie nie posiada defektów. Testowanie zmniejsza prawdopodobieństwo występowania w oprogramowaniu niewykrytych defektów, ale nawet jeżeli nie zostały znalezione żadne usterki, nie stanowi to dowodu poprawności oprogramowania.

Zasada 2 - Testowanie gruntowne jest niewykonalne

Przetestowanie wszystkiego (wszystkich kombinacji wejść i warunków początkowych) jest wykonalne tylko w trywialnych przypadkach. Zamiast testowania gruntownego, do kierowania testami należy wykorzystać analizę ryzyka i priorytetyzację.

Zasada 3 - Wczesne testowanie

Po to żeby wykryć usterki jak najwcześniej, testowanie rozpoczyna się w cyklu rozwoju oprogramowania lub systemu tak wcześnie jak to tylko jest możliwe i jest nakierowane na spełnienie zdefiniowanych celów.

Zasada 4 - Kumulowanie się błędów

Pracochłonność testowania jest dzielona proporcjonalnie do spodziewanej oraz zaobserwowanej gęstości błędów w modułach. Niewielka liczba modułów zwykle zawiera większość usterek znalezionych przez wydaniem lub jest odpowiedzialna za większość awarii na produkcji.

Zasada 5 - Paradoks pestycydów

Jeżeli te same testy są powtarzane bez zmian, to w końcu przestaną znajdować nowe usterki. Żeby przezwyciężyć "paradoks pestycydów", testy muszą być regularnie przeglądane i uaktualniane. Nowe, odmienne przypadki testowe muszą być projektowane w celu przetestowania różnych części oprogramowania lub systemu, żeby umożliwić odszukanie nowych usterek.

Zasada 6 - Testowanie zależy od kontekstu

Testowanie wykonuje się w różny sposób w różnym kontekście. Na przykład oprogramowanie krytyczne ze względu na bezpieczeństwo testuje się inaczej niż sklep internetowy.

Zasada 7 - Mylne przekonanie o braku błędów

Znajdowanie i naprawa usterek nie pomoże, jeżeli produkowany system nie nadaje się do użytkowania lub nie spełnia wymagań i oczekiwań użytkownika.

1.4 Podstawowy proces testowy

K1 35min

Pojęcia
testowanie potwierdzające, retesty, kryteria wyjścia, incydent, testowanie regresywne, podstawa testów, warunek testowy, pokrycie testowe, dane testowe, wykonanie testów, log testowy, plan testów, procedura testowa, polityka testów, zestaw testów, sumaryczny raport z testów, testalia

Wstęp

Najbardziej widocznym elementem testowania jest wykonywanie testów. Jednak, żeby testy były skuteczne i efektywne, plany testów powinny również uwzględniać czas potrzebny na zaplanowanie testów, zaprojektowanie przypadków testowych, przygotowanie do ich wykonania oraz ocenę wyników.

Podstawowy proces testowy składa się z następujących czynności:

- planowanie i nadzór nad testami
- analiza i projektowanie testów
- implementacja i wykonanie testów
- ocena kryteriów zakończenia i raportowanie
- czynności zamykające test

Chociaż wyżej wymienione czynności układają się w logiczny ciąg, to w praktyce mogą się zazębiać lub występować jednocześnie. Zwykle konieczne jest ich dostosowanie do kontekstu systemu i projektu.

K1

1.4.1 Planowanie i nadzór

Planowanie testów polega na zdefiniowaniu celów testowania i określeniu czynności testowych potrzebnych do wypełnienia misji i celów testowania.

Nadzór nad testami jest ciągłą aktywnością polegającą na porównywaniu postępów testów z planem oraz raportowaniu statusu (włącznie z odchyleniami od planu). Polega również na

podejmowaniu działań koniecznych do wypełnienia misji i celów projektu. Żeby nadzorować testy, zadania testowe trzeba monitorować przez cały projekt. Podczas planowania testów bierze się pod uwagę informacje pochodzące z monitorowanie i nadzoru testów.

Zadania związane z planowaniem i nadzorem nad testami zdefiniowane są w rozdziale 5 tego sylabusa.

K1

1.4.2 Analiza i projektowanie testów

Podczas analizy i projektowania testów ogólne cele testowania przekształcane są w konkretne warunki testowe i przypadki testowe.

Głównymi zadaniami analizy i projektowania testów są:

- przeglądanie podstawy testów (wymagań, poziomu integralności oprogramowania[2] (poziomu ryzyka), raportów analizy ryzyka, architektury, projektu, specyfikacji interfejsów)
- ocena testowalności podstawy testów i przedmiotu testów
- identyfikacja i priorytetyzacja warunków testowych na podstawie analizy elementów testowych, specyfikacji, zachowania i struktury oprogramowania
- projektowanie i priorytetyzacja przypadków testowych wysokiego poziomu
- ustalenie jakie dane testowe są potrzebne dla warunków testowych oraz przypadków testowych
- projektowanie środowiska testowego oraz identyfikacja potrzebnej infrastruktury i narzędzi
- tworzenie dwukierunkowego śledzenia pomiędzy podstawą testów oraz przypadkami testowymi

K1

1.4.3 Implementacja i wykonanie testów

Implementacja i wykonanie testów, to czynność, podczas której specyfikowane są procedury i skrypty testowe przez ustawienie przypadków testowych w konkretnej kolejności oraz dołączenie innych informacji potrzebnych do wykonania testów, konfigurowane jest środowisko testowe oraz wykonywane są testy.

Głównymi zadaniami implementacji i wykonania testów są:

- dokończenie, implementacja i priorytetyzacja przypadków testowych (włącznie z identyfikacją danych testowych)
- przygotowanie[3] i priorytetyzacja procedur testowych, tworzenie danych testowych oraz (opcjonalnie) przygotowywanie jarzm testowych i pisanie automatycznych skryptów testowych
- tworzenie zestawów testów z procedur testowych w celu efektywniejszego wykonania testów
- sprawdzenie, czy środowisko testowe zostało poprawnie skonfigurowane

- weryfikacja oraz uaktualnienie dwukierunkowego śledzenia pomiędzy podstawą testów oraz przypadkami testowymi
- wykonanie procedur testowych w zaplanowanej kolejności, ręcznie lub przy pomocy narzędzi do wykonywania testów
- zapisywanie wyników wykonania testów oraz zapisywanie identyfikatorów i wersji testowanego oprogramowania, narzędzi testowych oraz testaliów
- porównywanie wyników rzeczywistych z wynikami oczekiwanymi
- raportowanie rozbieżności jako incydentów oraz ich analiza w celu ustalenia ich przyczyny (usterki w kodzie, w danych testowych, w dokumencie testowym, albo pomyłka w trakcie wykonywania testów)
- powtarzanie czynności testowych jako rezultat akcji podjętych po stwierdzeniu rozbieżności, na przykład powtórne wykonanie testów niezaliczonych, aby potwierdzić naprawę defektu (testowanie potwierdzające), wykonanie poprawionych testów lub wykonanie testów w celu sprawdzenia czy w nie zmienianych częściach oprogramowania nie pojawiły się usterki lub czy naprawa usterek nie ujawniła innych defektów (testowanie regresywne).

K1

1.4.4 Ocena spełnienia kryteriów zakończenia i raportowanie

Ocena spełnienia kryteriów zakończenia polega na ocenie wykonania testów zgodnie z przyjętymi celami testowania. Powinna ona być wykonywana dla każdego poziomu testowania (patrz podrozdział 2.2).

Głównymi zadaniami oceny spełnienia kryteriów zakończenia są:

- sprawdzanie w logach (dziennikach) testów, czy zostały spełnione kryteria zakończenia testów określone podczas planowania
- ocenienie, czy potrzeba więcej testów lub czy nie powinny zostać zmienione kryteria zakończenia
- napisanie raportu podsumowującego testy dla interesariuszy

K1

1.4.5 Czynności zamykające testy

W ramach czynności zamykających testy zbierane są dane z zakończonych czynności testowych, żeby utrwalić doświadczenie, testalia, fakty i liczby. Czynności zamykające testy wykonywane są przy kamieniach milowych projektu takich jak: wydanie systemu, zakończenie (lub anulowanie) projektu testowego, osiągnięcie kamienia milowego, lub zakończenie wydania serwisowego.

Głównymi zadaniami wykonywanymi w ramach czynności zamykających testy są:

- sprawdzenie, które planowane produkty zostały dostarczone

- zamknięcie raportów incydentów lub utworzenie zgłoszeń zmian[4] dla tych, które pozostały otwarte
- udokumentowanie akceptacji systemu
- dokończenie i zarchiwizowanie testaliów, środowiska testowego i infrastruktury testowej do ponownego użycia w późniejszym terminie
- przekazanie testaliów do zespołu serwisowego
- przeanalizowanie doświadczeń by ustalić, jakie zmiany są potrzebne w przyszłych wydaniach i projektach
- wykorzystanie zebranych informacji do podniesienia dojrzałości testowania

1.5 Psychologia testowania

K2 25min

Pojęcia
zgadywanie błędów, niezależność testowania

Wstęp

Sposób myślenia stosowany podczas testowania i przeglądania różni się od stosowanego podczas rozwoju oprogramowania. Programiści posiadający odpowiednie nastawienie są w stanie testować swój kod, ale zwykle odpowiedzialność za testowanie przekazuje się testerom żeby wzmocnić koncentrację wysiłków oraz uzyskać dodatkowe korzyści, takie jak niezależne spojrzenie wyszkolonych, zawodowych testerów. Testowanie niezależne może być wykonywane na dowolnym poziomie testów.

Niezależność do pewnego stopnia jest często bardziej skuteczna w znajdowaniu defektów i awarii (unika się stronniczości autora). Nie może ona jednak zastąpić znajomości rzeczy i z tego względu programiści są w stanie efektywnie znajdować usterki w swoim własnym kodzie. Można zdefiniować kilka poziomów niezależności (od najniższego do najwyższego):

- testy zaprojektowane przez osobę, która napisała testowane oprogramowanie (niski poziom niezależności)
- testy zaprojektowane przez inną osobę (np. z zespołu programistów)
- testy zaprojektowane przez osobę z innego zespołu w organizacji (np. niezależnego zespołu testerów) lub przez specjalistów od testów (np. testów wydajnościowych lub użyteczności)
- testy zaprojektowane przez osobę z innej organizacji lub firmy (np. testy zlecone lub certyfikacja przez niezależny organ certyfikujący)

Ludzie i projekty kierują się celami. Ludzie mają skłonność do dopasowywania planów do celów określonych przez kierownictwo lub innych interesariuszy. Na przykład, żeby znajdować błędy albo żeby potwierdzić, że oprogramowania spełnia zadane cele. Dlatego bardzo ważne jest jasne wyrażenie celów testowania.

Znajdowanie błędów podczas testów może być odbierane jako krytykowanie produktu lub jego autora. Z tego powodu testowanie często jest postrzegane jako działanie destrukcyjne, nawet jeżeli jest bardzo konstruktywne z punktu widzenia zarządzania ryzykiem

produktowym. Wyszukiwanie awarii w systemie wymaga ciekawości, profesjonalnego pesymizmu, krytycznego spojrzenia, przywiązywania wagi do szczegółów, dobrej komunikacji z programistami oraz doświadczenia, na którym można oprzeć zgadywanie błędów.

Można uniknąć spięć pomiędzy testerami, a analitykami, projektantami i programistami przez komunikowanie błędów, usterek i awarii w sposób konstruktywny. Ta zasada ma zastosowanie zarówno do defektów znalezionych podczas przeglądów, jak i w testowaniu.

Tester i lider testów potrzebują dużych umiejętności interpersonalnych, żeby przekazywać fakty dotyczące usterek, postępów i ryzyka w sposób konstruktywny. Informacje na temat usterek w oprogramowaniu lub dokumencie mogą pomóc ich autorom w podnoszeniu kwalifikacji. Usterki znalezione i naprawione podczas testowania oszczędzają czas i pieniądze w przyszłości, a także ograniczają ryzyko.

Problemy z komunikacją mogą wystąpić jeżeli testerzy są postrzegani jedynie jako posłańcy przynoszący niechciane informacje o usterkach. Istnieje kilka sposobów na poprawienie komunikacji i relacji pomiędzy testerami i resztą zespołu:

- zacznij od współpracy, a nie od wojny - przypomnij wszystkim, że celem jest wyprodukowanie systemów o lepszej jakości
- komunikuj informacje na temat produktu w sposób neutralny, skoncentrowany na faktach bez krytykowania autora produktu, na przykład pisz obiektywne i konkretne raporty incydentów oraz uwagi z przeglądów
- spróbuj zrozumieć, co druga osoba czuje i dlaczego reaguje tak jak reaguje
- upewnij się, że druga strona zrozumiała, co powiedziałeś i upewnij się, że rozumiesz uwagi drugiej strony

1.6 Kodeks etyczny

K2 10min

Pojęcia
-

Zaangażowanie w testy pozwala poszczególnym testerom na poznanie poufnych oraz niejawnych informacji. Konieczny jest kodeks etyczny, który zapewni między innymi, że informacje te nie zostaną niewłaściwie użyte. Pamiętając o kodeksach etycznych dla inżynierów opublikowanych przez ACM i IEEE, ISTQB ogłasza następujący kodeks

- interes publiczny - certyfikowani testerzy oprogramowania postępują zgodnie z interesem publicznym
- klient i pracodawca - certyfikowani testerzy oprogramowania postępują zgodnie z najlepiej pojmowanym interesem swoich klientów i pracodawców, zgodnym z interesem publicznym
- produkt - certyfikowani testerzy oprogramowania dokładają wszelkich starań żeby dostarczane przez nich produkty (związane z testowanymi przez nich produktami i systemami) spełniały najwyższe standardy zawodowe

- osąd - certyfikowani testerzy oprogramowania są w swoich osądach uczciwi i niezależni
- kierownictwo - certyfikowani kierownicy testów postępują etycznie i promują etyczne podejście do kierowania testami
- zawód - certyfikowani testerzy oprogramowania promują uczciwość oraz reputację zawodu testera w zgodzie z interesem publicznym
- współpracownicy - certyfikowani testerzy oprogramowania są uczciwi wobec swoich współpracowników i wspierają ich oraz promują współpracę z deweloperami
- ja - certyfikowani testerzy oprogramowania uczą się przez całe życie swojego zawodu oraz promują etyczne podejście do praktyki zawodowej

Literatura

1.1.5 Black, 2001, Kaner, 2002

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

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

1.4 Hetzel, 1998

1.4.5 Black, 2001, Craig, 2002

1.5 Black, 2001, Hetzel, 1998

Przypisy

1. pass - tak jest przetłumaczone w słowniku SJSI
2. stopień w którym oprogramowanie spełnia lub musi spełniać zbiór określonych przez interesariuszy atrybutów programowych lub sprzętowych (np. złożoność oprogramowania, ocena ryzyka, poziom bezpieczeństwa, poziom zabezpieczeń, pożądana wydajność, niezawodność lub koszt), które zostały zdefiniowane żeby odzwierciedlać wagę oprogramowania dla interesariuszy.
3. development
4. change records

2. Testowanie w cyklu życia oprogramowania

K2 115min

Cele nauczania dla testowania w cyklu życia oprogramowania.

Te cele określają co będziesz potrafił po zakończeniu modułu.

2.1 Modele wytwarzania oprogramowania K2

LO-2.1.1 Kandydat potrafi wyjaśnić powiązania pomiędzy rozwojem oprogramowania, czynnościami testowymi oraz produktami w cyklu życia oprogramowania i podać przykłady na podstawie cech projektu i produktu oraz kontekstu. (K2)
LO-2.1.2 Kandydat akceptuje fakt, że modele wytwarzania oprogramowania muszą zostać zaadaptowane do cech projektu i produktu. (K1)
LO-2.1.3 Kandydat pamięta atrybuty dobrego testowania mające zastosowanie w każdym z modeli życia oprogramowania. (K1)

2.2 Poziomy testów K2

LO-2.2.1 Kandydat potrafi porównać różne poziomy testów: główne cele, typowe przedmioty testów, typowe cele testowania (np. strukturalne lub funkcjonalne) i związane z nimi produkty, testerów, typy usterek i awarii do znalezienia. (K2)

2.3 Typy testów K2

LO-2.3.1 Kandydat potrafi porównać podając przykłady cztery typy testów (funkcjonalne, niefunkcjonalne, strukturalne oraz związane ze zmianami). (K2)
LO-2.3.2 Kandydat uznaje, że testy funkcjonalne i strukturalne występują na każdym poziomie testów. (K1)
LO-2.3.3 Kandydat potrafi wymienić i opisać różne typy testów niefunkcjonalnych bazujących na wymaganiach niefunkcjonalnych. (K2)
LO-2.3.4 Kandydat potrafi wymienić i opisać typy testów bazujące na analizie struktury lub architektury systemu. (K2)
LO-2.3.5 Kandydat potrafi opisać cel wykonywania testów potwierdzających i regresywnych. (K2)

2.4 Testowanie pielęgnacyjne K2

LO-2.4.1 Kandydat potrafi porównać testy pielęgnacyjne (testowanie istniejącego systemu) do testowania nowej aplikacji uwzględniając typy testów, powody rozpoczęcia[1] testowania oraz ilość testów. (K2)
LO-2.4.2 Kandydat potrafi wymienić powody wykonywania testów pielęgnacyjnych (zmiany, migracja i wycofanie systemu). (K1)
LO-2.4.3 Kandydat potrafi opisać rolę testów regresywnych i analizy wpływu w utrzymaniu. (K2)

2.1 Modele wytwarzania oprogramowania

K2 20min

Pojęcia
komercyjne oprogramowanie z półki (COTS), iteracyjno-przyrostowy model wytwarzania, walidacja, weryfikacja, model V

Wstęp

Testowanie nie dzieje się w izolacji. Czynności testowania powiązane są z działaniami związanymi z rozwojem oprogramowania. Różne modele wytwarzania oprogramowania wymagają różnego podejścia do testowania.

K2

2.1.1 Model V (model sekwencyjny)

Najczęściej spotykany model V posiada cztery poziomy testowania odpowiadające czterem poziomom rozwoju oprogramowania. Istnieją też inne warianty tego modelu.

Cztery poziomy testowania używane w tym sylabusie to:

- testy modułowe (jednostkowe)
- testy integracyjne
- testy systemowe
- testy akceptacyjne

W praktyce model V może posiadać więcej, mniej lub w ogóle inne poziomy testowania i rozwoju oprogramowania, zależy to od konkretnego projektu i wytwarzanego oprogramowania. Na przykład po testach modułowych może następować testowanie integracji modułów, a po testach systemowych - testy integracji systemów.

Produkty związane z wytwarzaniem oprogramowania (takie jak scenariusze biznesowe lub przypadki użycia, wymagania, projekt i kod źródłowy) są często podstawą do testowania na jednym lub wielu poziomach. Odniesienia do ogólnych produktów znajdują się w Capability Maturity Model Integration (CMMI) oraz "Software life cycle processes" (IEEE/IEC 12207). Podczas wytwarzania tych produktów można już wykonywać ich weryfikację i walidację (oraz wstępne projektowanie testów).

K2

2.1.2 Modele iteracyjno-przyrostowe

Iteracyjno-przyrostowe wytwarzanie oprogramowania to proces zbierania wymagań, projektowania, budowania oraz testowania systemu zorganizowany w krótsze cykle rozwojowe. Na przykład: prototypowanie, RAD, Rational Unified Process (RUP) oraz metodyki zwinne. System wyprodukowany według takiego modelu może być przetestowany na kilku poziomach w każdej iteracji. Przyrost, dodany do innych wytworzonych wcześniej,

staje się rosnącym systemem częściowym, który również powinien być przetestowany. Testowanie regresywne jest bardzo ważne w każdej iteracji oprócz pierwszej. Każdy przyrost może podlegać zarówno weryfikacji jak i walidacji.

K2

2.1.3 Testowanie w cyklu życia oprogramowania

W każdym modelu rozwoju oprogramowania dobre testowanie posiada kilka niezmiennych cech:

- dla każdej czynności związanej z wytworzeniem oprogramowania istnieją odpowiadające jej czynności związane z testowaniem
- każdy poziom testowania ma zdefiniowane cele
- analiza i projektowanie testów dla danego poziomu powinny rozpoczynać się już podczas odpowiadającej im fazy wytwarzania
- testerzy powinni uczestniczyć w przeglądach już od wczesnych wersji dokumentacji tworzonej podczas wytwarzania

Poziomy testowania mogą być łączone lub organizowane na różne sposoby w zależności od natury projektu lub architektury systemu. Na przykład przy integracji oprogramowania z półki w jeden system nabywca może przeprowadzić testy integracyjne na poziomie systemowym (np. integracja z infrastrukturą i innymi systemami lub wdrożenie systemu) oraz testy akceptacyjne (funkcjonalne i niefunkcjonalne oraz testy akceptacyjne użytkownika i testy produkcyjne).

2.2 Poziomy testów

K2 40min

Pojęcia
testowanie alfa, testowanie beta, testy modułowe, sterownik, testowanie w warunkach polowych, wymaganie funkcjonalne, integracja, testy integracyjne, wymaganie niefunkcjonalne, testowanie odporności, zaślepka, testy systemowe, środowisko testowe, poziom testów, wytwarzanie sterowane testami, testowanie akceptacyjne przez użytkownika

Wstęp

Dla każdego poziomu testowania można zdefiniować: ogólne cele, produkty, na podstawie których tworzy się przypadki testowe (podstawę testów), przedmiot testów (to co jest testowane), typowe defekty i awarie do wykrycia, wymagania na jarzmo testowe oraz wsparcie narzędziowe, środowisko testowe, specyficzne podejście, odpowiedzialność.

Podczas planowania testów powinno zostać uwzględnione również testowanie danych konfiguracyjnych.

2.2.1 Testy modułowe

Podstawa testów: K2

- wymagania na moduły
- projekt szczegółowy
- kod

Typowe obiekty testów:

- moduły
- programy
- programy do konwersji lub migracji danych
- moduły bazodanowe

Testy modułowe polegają na wyszukiwaniu błędów i weryfikacji funkcjonalności oprogramowania (np. modułów, programów, obiektów, klas), które można testować oddzielnie. Może być wykonywane w izolacji od reszty systemu, w zależności od kontekstu cyklu rozwoju oprogramowania i od samego systemu. Można podczas nich użyć zaślepek, sterowników testowych oraz symulatorów.

Testy modułowe mogą zawierać testy funkcjonalności oraz niektórych atrybutów niefunkcjonalnych, takich jak stopień wykorzystania zasobów (np. wycieków pamięci) lub odporności, jak również testy strukturalne (np. pokrycia decyzji). Przypadki testowe są projektowane na podstawie takich produktów jak specyfikacja modułu, projekt oprogramowania lub model danych.

Testy modułowe zwykle wykonuje się mając dostęp do kodu źródłowego i przy wsparciu środowiska rozwojowego (np. bibliotek do testów jednostkowych, narzędzi do debagowania) oraz, w praktyce, zwykle angażują też programistę, który jest autorem kodu. Usterki są usuwane jak tylko zostaną wykryte, bez formalnego zarządzania nimi.

Jednym z podejść do testów modułowych jest przygotowanie i zautomatyzowanie przypadków testowych przed kodowaniem. Podejście to nazywane jest "najpierw testuj" lub wytwarzanie sterowane testowaniem. Jest to podejście wysoce iteracyjne i opiera się na cyklach tworzenia przypadków testowych, a następnie budowaniu i integracji niewielkich fragmentów kodu, wykonywaniu testów modułowych, poprawianiu usterek i powtarzaniu tego procesu aż testy zostaną zaliczone.

K2

2.2.2 Testy integracyjne

Podstawa testów:

- projekt oprogramowania i systemu
- architektura
- przepływy procesów
- przypadki użycia

Typowe obiekty testów:

- implementacja baz danych podsystemów
- infrastruktura
- interfejsy
- konfiguracja systemu i dane konfiguracyjne

Testy integracyjne sprawdzają interfejsy pomiędzy modułami, interakcje z innymi częściami systemu (takimi jak system operacyjny, system plików i sprzęt) oraz interfejsy pomiędzy systemami.

Testy integracyjne mogą być wykonywane na więcej niż jednym poziomie i dla przedmiotów testów o różnej wielkości:

- testowanie integracji modułów sprawdza interakcje pomiędzy modułami oprogramowania i jest wykonywane po testach modułowych
- testowanie integracji systemów sprawdza interakcje pomiędzy różnymi systemami lub pomiędzy sprzętem a oprogramowaniem i może być wykonywane po testach systemowych

  W takim przypadku organizacja rozwijająca system może kontrolować tylko jedną stronę interfejsu. Może to zostać uznane za ryzyko. Procesy biznesowe zaimplementowane jako przepływ pracy[2] mogą angażować wiele systemów. W takim przypadku istotne mogą okazać się różnice między platformami, na których zaimplementowane są poszczególne systemy.

Im większy jest zakres integracji, tym trudniejsze może być określenie, który moduł lub system zawiera defekt co powoduje zwiększone ryzyko i dłuższy czas rozwiązywania problemów.

Systematyczne strategie integracji mogą bazować na architekturze systemu (np. strategie wstępująca i zstępująca), zadaniach funkcjonalnych, sekwencjach przetwarzania transakcji albo na innym aspekcie systemu lub modułu. Żeby ułatwić namierzanie usterek oraz ich wczesne wykrywanie, w normalnych warunkach integracja powinna być raczej prowadzona metodą inkrementalną niż metodą "wielkiego wybuchu".

Podczas testów integracyjnych można wykonać testy niektórych atrybutów niefunkcjonalnych (np. wydajności) na równi z testami funkcjonalnymi.

Na każdym etapie integracji, testerzy koncentrują się wyłącznie na samej integracji. Na przykład, gdy integrują moduł A z modułem B, interesują się tylko testowaniem komunikacji pomiędzy modułami, a nie funkcjonalnością poszczególnych modułów, gdyż ta była sprawdzona wcześniej w testach modułowych. Można tu wykorzystać zarówno podejście funkcjonalne jak i strukturalne.

W idealnym przypadku tester powinien rozumieć architekturę i mieć wpływ na planowanie integracji. Jeżeli testy integracyjne planowane są zanim moduły lub systemy zostały

wyprodukowane, można ich rozwój ustawić w kolejności pozwalającej na najbardziej efektywne testowanie.

K2

2.2.3 Testy systemowe

Podstawa testów:

- wymagania na system i oprogramowanie
- przypadki użycia
- specyfikacja funkcjonalna
- raporty z analizy ryzyka

Typowe obiekty testów:

- podręczniki systemowe, użytkownika i operacyjne
- konfiguracje systemu i dane konfiguracyjne

Testy systemowe zajmują się zachowaniem systemu/produktu. Zakres testów powinien być jasno określony w głównym planie testów oraz w planach testów poszczególnych poziomów.

Środowisko testowe, podczas testów systemowych, powinno być zgodne ze środowiskiem docelowym/produkcyjnym w jak najwyższym możliwym stopniu, żeby zminimalizować ryzyko wystąpienia awarii spowodowanych przez środowisko, które nie zostałyby wykryte podczas testów.

Testy systemowe mogą zawierać testy oparte na ryzyku lub wymaganiach, procesie biznesowym, przypadkach użycia lub jeszcze innych wysokopoziomowych opisach słownych lub modelach zachowania systemu, interakcji z systemem operacyjnym i zasobami systemowymi.

Testy systemowe powinny sprawdzać funkcjonalne jak i niefunkcjonalne wymagania na system oraz jakość danych. Wymagania mogą być wyrażone w formie tekstu lub modeli. Testerzy muszą umieć sobie poradzić z wymaganiami niekompletnymi lub nieudokumentowanymi. Testowanie systemowe wymagań funkcjonalnych rozpoczyna się przez użycie najbardziej odpowiednich technik opartych na specyfikacji (czarnoskrzynkowych) dla testowanego aspektu systemu. Na przykład można utworzyć tablicę decyzyjną zawierającą kombinacje skutków opisane w regułach biznesowych. Następnie można użyć technik opartych na strukturze (białoskrzynkowych) do oceny dokładności testowania w odniesieniu do elementów struktury, takich jak menu lub nawigacja po stronach webowych. (Patrz rozdział 4)

Testy systemowe są często wykonywane przez niezależny zespół testowy.

2.2.4 Testy akceptacyjne

Podstawa testów: K2

- wymagania użytkownika
- wymagania systemowe
- przypadki użycia
- procesy biznesowe
- raporty z analizy ryzyka

Typowe obiekty testów:

- proces biznesowy na systemie w pełni zintegrowanym
- procesy utrzymania i obsługi
- procedury pracy użytkowników
- formularze
- raporty
- dane konfiguracyjne

Odpowiedzialność za testy akceptacyjne leży często po stronie klientów lub użytkowników systemu. Mogą w nie być zaangażowani również inni interesariusze.

Celem testów akceptacyjnych jest nabranie zaufania do systemu, jego części lub pewnych atrybutów niefunkcjonalnych. Wyszukiwanie usterek nie jest tym, na czym skupiają się testy akceptacyjne. Mogą one oceniać gotowość systemu do wdrożenia i użycia, chociaż nie muszą być ostatnim poziomem testowania. Na przykład może po nich następować testowanie integracji systemów w większej skali.

Testy akceptacyjne mogą pojawić się w wielu momentach cyklu życia oprogramowania. Na przykład:

- oprogramowanie z półki może podlegać testom akceptacyjnym, gdy jest instalowane lub integrowane
- testy akceptacyjne użyteczności modułu mogą być wykonane w czasie testów modułowych
- testy akceptacyjne nowej funkcjonalności mogą być przeprowadzone przed testami systemowymi

Typowymi formami testów akceptacyjnych są:

Testowanie akceptacyjne przez użytkownika

Zwykle sprawdza przydatność[3] systemu dla użytkowników.

(Akceptacyjne) testy produkcyjne

Akceptacja systemu przez administratorów:

- testowanie wykonywania i odtwarzania kopii zapasowych
- uruchamianie systemu po awarii[4]
- zarządzanie użytkownikami
- zadania związane z utrzymaniem systemu
- ładowanie danych i inne zadania związane z migracją
- okresowe sprawdzenie słabych punktów zabezpieczeń

Testy akceptacyjne zgodności z umową i testy zgodności legislacyjnej

Testy zgodności z umową są wykonywane przez sprawdzenie spełnienia kryteriów akceptacji zapisanych w kontrakcie na wykonanie oprogramowania na zamówienie. Te kryteria akceptacji powinny być zdefiniowane w momencie negocjacji umowy. Testy zgodności legislacyjnej wykonuje się sprawdzając, czy oprogramowanie jest zgodne z wszystkimi przepisami prawnymi, z którymi musi być zgodne, takimi jak rozporządzenia rządowe, inne akty prawne lub przepisy dotyczące bezpieczeństwa.

Testy alfa i beta (lub testy w warunkach polowych)

Producenci oprogramowania tworzonego na szeroki rynek (oprogramowanie z półki) często chcą uzyskać opinię potencjalnych lub obecnych klientów, zanim oprogramowanie zostanie wypuszczone do sprzedaży. Testy alfa są wykonywane u producenta, ale nie przez zespół projektowy. Testy beta, lub testy polowe, wykonywane są przez klientów lub potencjalnych klientów w ich własnych lokalizacjach.

W różnych organizacjach mogą być w użyciu inne nazwy, takie jak fabryczne testy akceptacyjne lub docelowe testy akceptacyjne[5] w sytuacji, gdy system jest testowany przed i po zainstalowaniu u klienta.

2.3 Typy testów

K2 40min

Pojęcia
testowanie czarnoskrzynkowe, pokrycie kodu, testowanie funkcjonalne, testowanie współdziałania, testowanie obciążeniowe, testowanie pielęgnowalności, testowanie wydajnościowe, testowanie przenaszalności, testowanie niezawodności, testowanie zabezpieczeń, testowanie przeciążające, testowanie strukturalne, testowanie użyteczności, testowanie białoskrzynkowe

Wstęp

Grupa czynności testowych może być nakierowana na weryfikację systemu (lub części systemu) bazując na konkretnym powodzie lub celu testów.

Testy określonego typu skupiają się na konkretnym celu testu, którym może być którekolwiek z poniższych:

- przetestowanie jakiejś funkcji wykonywanej przez oprogramowanie

- przetestowanie jakiegoś niefunkcjonalnego atrybutu jakościowego (takiego jak niezawodność lub użyteczność)
- przetestowanie struktury lub architektury systemu
- testy związane ze zmianami, to jest potwierdzaniem, że usterki zostały naprawione (testowanie potwierdzające) i szukaniem niezamierzonych zmian (testowanie regresywne).

Można opracować model oprogramowania i użyć go w testach strukturalnych (np. model przepływu sterowania, model struktury menu), testach niefunkcjonalnych (np. model wydajności, model użyteczności, model zagrożeń zabezpieczeń) lub funkcjonalnych (np. model przepływu procesu, model przejść między stanami lub specyfikacja w języku naturalnym).

K2

2.3.1 Testowanie funkcji (testowanie funkcjonalne)

Funkcje, jakie ma pełnić system, podsystem lub moduł, mogą być opisane w produktach takich jak specyfikacja wymagań, przypadki użycia lub specyfikacja funkcjonalna. Mogą też być nieudokumentowane. Funkcje są tym "co" system robi.

Testy funkcjonalne dotyczą funkcji lub innych cech[6] (opisanych w dokumentach lub domniemanych przez testerów) oraz ich współdziałania z innymi systemami. Można je wykonywać na wszystkich poziomach (np. testy modułowe mogą bazować na specyfikacji modułów).

Do tworzenia warunków testowych oraz przypadków testowych dla funkcjonalności systemu mogą zostać użyte techniki oparte na specyfikacji (Patrz rozdział 4). Testy funkcjonalne zajmują się zewnętrznym zachowaniem oprogramowania (testy czarnoskrzynkowe).

Jeden z typów testów funkcjonalnych - testowanie zabezpieczeń - sprawdza funkcje (np. zapory) pozwalające na wykrycie zagrożeń, takich jak wirusy, pochodzących od złośliwych obcych. Inny typ testów funkcjonalnych: testowanie współdziałania, ocenia zdolność oprogramowania do współpracy z jednym lub większą liczbą wskazanych modułów lub systemów.

K2

2.3.2 Testowanie atrybutów niefunkcjonalnych (testowanie niefunkcjonalne)

Testowanie niefunkcjonalne obejmuje następujące (ale nie tylko te) typy testów: testowanie wydajnościowe, testowanie obciążeniowe, testowanie przeciążeniowe, testowanie użyteczności, testowanie pielęgnowalności, testowanie niezawodności oraz testowanie przenaszalności. Testowanie niefunkcjonalne polega na sprawdzeniu "jak" system działa.

Testowanie niefunkcjonalne może być wykonywane na wszystkich poziomach testów. Termin testy niefunkcjonalne oznacza testy wymagane do zmierzenia właściwości systemów i oprogramowania, które mogą zostać określone ilościowo na zmiennej skali, takich jak czasy odpowiedzi w testach wydajnościowych. Testy te mogą zostać odniesione do modelu jakości

oprogramowania np. "Software Engineering – Software Product Quality" (ISO 9126). Testy niefunkcjonalne zajmują się zewnętrznym zachowaniem oprogramowania i w większości wypadków wykorzystują techniki czarnoskrzynkowe.

K2

2.3.3 Testowanie struktury/architektury oprogramowania (testowanie strukturalne)

Testy strukturalne (białoskrzynkowe) można wykonywać na każdym poziomie testowania. Technik strukturalnych najlepiej użyć po technikach opartych na specyfikacji, po to by zmierzyć dokładność testowania przez ocenę stopnia pokrycia wybranego typu struktury.

Pokrycie, to stopień, w jakim struktura została przetestowana przez zestaw testów wyrażony jako odsetek pokrytych elementów. Jeżeli pokrycie jest niższe niż 100%, można zaprojektować więcej testów, żeby przetestować te elementy, które zostały pominięte, i w ten sposób zwiększyć pokrycie. Techniki oparte na pokryciu opisane są w rozdziale 4.

Do pomiaru pokrycia (na przykład decyzji lub instrukcji) na wszystkich poziomach, ale szczególnie na poziomie testów modułowych i poziomie testów integracji modułów, mogą zostać użyte narzędzia. Testowanie strukturalne może zostać oparte na architekturze systemu, na przykład hierarchii wywołań.

Podejście strukturalne może mieć zastosowanie również w testach systemowych, testach integracji systemów oraz testach akceptacyjnych (np. do modeli biznesowych lub struktur menu).

K2

2.3.4 Testowanie związane ze zmianami: testowanie potwierdzające oraz regresywne

Po wykryciu i naprawieniu defektu, powinien zostać wykonany retest, żeby potwierdzić usunięcie usterki. Takie testy nazywane są testami potwierdzającymi. Debagowanie (lokalizacja oraz poprawianie usterek) jest czynnością programistyczną, a nie testową.

Testowanie regresywne, to powtórzenie testów na już przetestowanym programie wykonywane po modyfikacjach żeby wykryć nowe usterki lub usterki odsłonięte na skutek wykonanych zmian. Usterki te mogą występować w testowanym oprogramowaniu, jak również w innych powiązanych lub niepowiązanych modułach. Testy regresywne wykonuje się po zmianach w oprogramowaniu, a także po zmianach w jego środowisku. Zakres testów regresywnych wynika z ryzyka nieznalezienia usterek w oprogramowaniu, które poprzednio działało poprawnie.

Testy, które mają być stosowane w testowaniu potwierdzającym i regresywnym muszą być powtarzalne.

Testy regresywne można wykonywać na wszystkich poziomach testów i dla wszystkich typów testów: funkcjonalnych, niefunkcjonalnych i strukturalnych. Zestawy testów regresywnych są wykonywane wiele razy i są stosunkowo niezmienne, wobec czego są dobrymi kandydatami do automatyzacji.

2.4 Testowanie pielęgnacyjne

K2 15min

Pojęcia
analiza wpływu, testowanie pielęgnacyjne

Wdrożony system często musi działać przez lata lub dekady. W tym czasie system, jego dane konfiguracyjne i jego środowisko są często poprawianie, zmieniane i rozszerzane. Dla dobrego testowania pielęgnacyjnego krytyczne jest planowanie wydań z wyprzedzeniem. Konieczne jest rozróżnianie pomiędzy planowanymi wydaniami i szybkimi poprawkami[7]. Testowanie pielęgnacyjne wykonuje się na działającym systemie na skutek modyfikacji, migracji lub złomowania oprogramowania lub systemu.

Modyfikacjami mogą być planowane ulepszenia (np. według planowych wydań), poprawki lub naprawy awaryjne oraz zmiany środowiska, takie jak planowane podniesienia wersji systemu operacyjnego, bazy danych lub oprogramowania z półki albo łaty zabezpieczeń systemu operacyjnego.

Testy pielęgnacyjne migracji oprogramowania (np. z jednej platformy na inną) powinny, oprócz testów zmian w oprogramowaniu, uwzględniać również testy produkcyjne nowego środowiska. Testy migracji (testowanie konwersji) są również potrzebne, gdy dane z innej aplikacji są migrowane do utrzymywanego systemu.

Testy pielęgnacyjne związane z wycofywaniem oprogramowania mogą zawierać testy migracji lub archiwizacji danych, jeżeli wymagany jest długi okres ich przechowywania.

Testy pielęgnacyjne, oprócz przetestowania tego co zostało zmienione, zawierają testy regresywne tych części systemu, które się nie zmieniły. Zakres testów pielęgnacyjnych zależy od ryzyka zmian, wielkości istniejącego systemu oraz zakresu zmian. W zależności od zmian, testowanie pielęgnacyjne może być wykonane na niektórych lub wszystkich poziomach testowania oraz dla wybranych lub wszystkich typów testów.

Określenie, jaki wpływ na istniejący system mogą mieć zmiany, nazywamy analizą wpływu. Stosuje się ją, aby ustalić zakres testów regresywnych. Analiza wpływu może zostać użyta do wybrania zestawu testów regresyjnych.

Testowanie pielęgnacyjne może być trudne do wykonania, jeżeli specyfikacja oprogramowania jest przestarzała lub gdy jej brak, lub gdy nie są dostępni testerzy posiadający wiedzę dziedzinową.

Literatura

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

2.2 Hetzel, 1998

2.2.4 Copeland, 2004, Myers, 1979

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

2.3.2 Black, 2001, ISO 9126

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

2.3.4 Hetzel, 1998, IEEE 829

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

Przypisy

1. triggers
2. workflow
3. fitness for use
4. disaster recovery
5. site acceptance testing
6. features
7. hot fix

-

3. Statyczne techniki testowania

K2 60min

Cele nauczania dla technik statycznych.

Te cele określają co będziesz potrafił po zakończeniu modułu.

3.1 Techniki statyczne a proces testowania (K2)

LO-3.1.1 Kandydat potrafi rozpoznać produkty, które mogą zostać sprawdzone przez różne techniki testowania statycznego. (K1)
LO-3.1.2 Kandydat potrafi opisać znaczenie i wartość statycznych technik testowania w ocenie produktów procesu rozwoju oprogramowania. (K2)
LO-3.1.3 Kandydat potrafi wyjaśnić różnice pomiędzy testami statycznymi i dynamicznymi. (K2)

3.2 Proces przeglądu (K2)

LO-3.2.1 Kandydat pamięta kroki, role i odpowiedzialności związane z typowym przeglądem formalnym. (K2)
LO-3.2.2 Kandydat potrafi wyjaśnić różnice pomiędzy różnymi typami przeglądów: przeglądem nieformalnym, przeglądem technicznym, przejrzeniem i inspekcją. (K2)
LO-3.2.3 Kandydat potrafi wskazać czynniki wpływające na skuteczne przeprowadzanie przeglądów. (K2)

3.3 Analiza statyczna przy pomocy narzędzi (K2)

LO-3.3.1 Kandydat pamięta typowe błędy wykrywane przez analizę statyczną i umie porównać je z przeglądami i testami dynamicznymi. (K1)
LO-3.3.2 Kandydat potrafi opisać używając przykładów typowe korzyści z analizy statycznej. (K1)
LO-3.3.3 Kandydat potrafi wymienić typowe defekty kodu i projektu, które mogą zostać wykryte przez narzędzia do analizy statycznej. (K1)

3.1 Techniki statyczne a proces testowania

K2 15min

Pojęcia
testowanie dynamiczne, testowanie statyczne

W przeciwieństwie do technik dynamicznych, które wymagają uruchomienia oprogramowania, techniki statyczne polegają na sprawdzeniu ręcznym (przeglądy) lub analizie automatycznej (analiza statyczna) kodu lub innych dokumentów projektowych bez uruchamiania kodu.

Przeglądy są jednym ze sposobów na testowanie produktów procesu wytwarzania oprogramowania (łącznie z kodem). Przeglądy można wykonywać na długo przed wykonaniem testów dynamicznych. Usterki znalezione podczas przeglądów we wczesnych fazach produkcji oprogramowania (np. defekty znalezione w wymaganiach) często okazują się dużo tańsze do usunięcia niż te wykryte podczas wykonywania testów dynamicznych.

Przeglądy można wykonywać całkowicie ręcznie, ale istnieje dla nich również wsparcie narzędziowe. Główną czynnością wykonywaną manualnie jest sprawdzenie produktu i zanotowanie uwag. Przeglądom mogą podlegać wszystkie produkty procesu wytwarzania oprogramowania: specyfikacja wymagań, projekt, kod, plany testów, specyfikacja testów, przypadki testowe, skrypty testowe, podręcznik użytkownika oraz strony webowe.

Główne korzyści z wykonywania przeglądów to: wczesne wykrycie i naprawa usterek, zwiększenie produktywności produkcji oprogramowania, redukcja czasu produkcji oprogramowania, zmniejszenie kosztów i czasu testowania, ogólne zmniejszenie kosztu wytwarzania i użytkowania oprogramowania, zmniejszenie liczby usterek oraz usprawnienie komunikacji. Przeglądy mogą wykrywać braki (na przykład w wymaganiach), które trudno jest wykryć w testach dynamicznych.

Przeglądy, analiza statyczna oraz testy dynamiczne mają ten sam cel – znajdowanie usterek. Te trzy techniki uzupełniają się wzajemnie, mogą skutecznie i efektywnie znajdować różne typy błędów. Techniki statyczne, w odróżnieniu od testów dynamicznych, znajdują raczej przyczyny awarii (usterki), a nie same awarie.

Do typowych usterek, które łatwiej wykryć w testach statycznych niż dynamicznych należą: odchylenia od standardów, usterki w wymaganiach, usterki w projekcie, niedostateczna pielęgnowalność oraz nieprawidłowe specyfikacje interfejsów.

3.2 Proces przeglądu

K2 25min

Pojęcia
kryteria wejścia, przegląd formalny, przegląd nieformalny, inspekcja, metryka, moderator, przegląd koleżeński, przeglądający, protokólant, przegląd techniczny, przejrzenie

Wstęp

Istnieją różne typy przeglądów, od nieformalnych, cechujących się brakiem spisanych instrukcji dla przeglądających, do systematycznych, uwzględniających przygotowanie zespołu, zapisanie wyników przeglądu oraz udokumentowane procedury wykonania przeglądu. Stopień sformalizowania przeglądu zależy od takich czynników jak dojrzałość procesu produkcyjnego, czy wymagania wynikające z prawa lub konieczność udokumentowania przebiegu przeglądu.

Proces przeprowadzenia przeglądu zależy od jego celów (np. znajdowanie usterek, zrozumienie, przekazanie wiedzy testerom i nowym członkom zespołu lub przedyskutowanie i podjęcia uzgodnionej decyzji).

3.2.1 Kroki przeglądu formalnego

Przegląd formalny zwykle składa się z następujących faz: K1

1. planowanie

- definiowanie kryteriów przeglądu
- wybór uczestników przeglądu
- przydział ról
- ustalenie kryteriów wejścia i zakończenia dla bardziej formalnych typów przeglądów (np. dla inspekcji)
- wybór fragmentów dokumentu do przejrzenia

2. rozpoczęcie

- rozesłanie dokumentów
- opisanie celów przeglądu, procesu i dokumentów uczestnikom przeglądu
- sprawdzenie kryteriów wejścia (dla bardziej formalnych typów przeglądów)

3. przygotowanie indywidualne

- przygotowanie przed spotkaniem przeglądowym przez przejrzenie dokumentów
- zapisywanie potencjalnych defektów, pytań i komentarzy

4. kontrola/ocena/zapisanie wyników (spotkanie przeglądowe)

- dyskusja lub spisywanie, z udokumentowaniem wyników i sporządzeniem protokołu (dla bardziej formalnych typów przeglądów)
- zapisywanie defektów i rekomendacji dotyczących ich poprawiania, podejmowanie decyzji co do defektów

5. poprawki

- naprawianie znalezionych defektów, zwykle wykonywane przez autora
- uaktualnianie statusów defektów (w przeglądach formalnych)

6. zakończenie

- sprawdzenie, że usterki zostały obsłużone
- zbieranie metryk
- sprawdzanie kryteriów zakończenia (dla bardziej formalnych typów przeglądów)

K1

3.2.2 Role i odpowiedzialność

Uczestnicy przeglądów formalnych mogą pełnić następujące role:

kierownik
: Decyduje o wykonaniu przeglądów, przydziela czas w harmonogramie projektu i sprawdza, czy zostały zrealizowane cele przeglądu

moderator
: Osoba, która prowadzi przegląd dokumentu lub zbioru dokumentów, włączając w to planowanie przeglądu, prowadzenie spotkań i sprawdzenie czy ustalenia ze spotkania zostały wypełnione. Jeżeli jest to konieczne, moderator może mediować pomiędzy różnymi punktami widzenia i często jest osobą, od której zależy sukces przeglądu.

autor
: Osoba która napisała lub nadzorowała powstanie dokumentu podlegającego przeglądowi.

przeglądający
: Osoby, z odpowiednim przygotowaniem biznesowym lub technicznym (również nazywani kontrolerami lub inspektorami), które po niezbędnym przygotowaniu, znajdują i spisują uwagi (np. usterki) do przeglądanego produktu. Przeglądający powinni zastać tak dobrani, żeby reprezentowali różne spojrzenia i różne role w procesie przeglądu. Przeglądający powinni brać udział w każdym spotkaniu przeglądowym.

protokólant
: Dokumentuje wszystkie zagadnienia, problemy lub różnice zdań, które pojawiły się na spotkaniu przeglądowym.

Przeglądanie produktów programistycznych lub innych produktów z nimi związanych z różnych perspektyw oraz wykorzystywanie list kontrolnych może sprawić, że przeglądy będą skuteczniejsze i bardziej efektywne. Na przykład listy kontrolne uwzględniające różne spojrzenia takie jak perspektywa użytkownika, serwisanta, testera lub operatora, albo lista kontrolna typowych problemów z wymaganiami mogą pomóc w wykrywaniu nie wykrytych wcześniej defektów.

K2

3.2.3 Typy przeglądów

Pojedynczy produkt programistyczny lub inny powiązany z nim produkt może podlegać więcej niż jednemu przeglądowi. Jeżeli wykorzystywane jest kilka typów przeglądów, to mogą być wykonywane w różnym porządku. Na przykład przegląd nieformalny może być przeprowadzony przed przeglądem technicznym, albo inspekcja specyfikacji wymagań może poprzedzać przejrzenie ich z klientem.

Głównymi cechami, opcjami i celami powszechnie stosowanych typów przeglądów są:

Przegląd nieformalny

- brak formalnego procesu
- może przybrać formę programowania w parach oraz przegląd projektu lub kodu przez kierownika zespołu
- może być udokumentowany
- jego użyteczność może być różna w zależności od przeglądających

- główny cel: tani sposób na osiągnięcie niewielkich korzyści

Przejrzenie

- spotkanie jest prowadzone przez autora
- może przybrać formę scenariuszy, uruchamiania "na sucho", grupa kolegów
- sesje nie ograniczone czasowo
    - opcjonalnie przygotowanie przeglądających przed spotkaniem
    - opcjonalnie raport z przeglądu, lista uwag
- opcjonalnie protokólant (którym nie jest autor)
- w praktyce może być od całkiem nieformalnego do bardzo formalnego
- główne cele: uczenie się, zrozumienie, znajdowanie usterek

Przegląd techniczny

- posiada zdefiniowany proces wykrywania defektów m.in. przez kolegów i ekspertów technicznych z opcjonalnym udziałem kierownictwa
- może być organizowany jako przegląd koleżeński bez udziału kierownictwa
- w idealnej sytuacji prowadzony przez przeszkolonego moderatora (nie autora)
- przygotowanie przeglądających przed spotkaniem
- opcjonalnie z wykorzystaniem list kontrolnych
- przygotowanie raportu z przeglądu, który zawiera listę uwag, ocenę czy produkt programistyczny spełnia wymagania i, tam gdzie jest to potrzebne, rekomendacje związane z uwagami
- w praktyce może być wykonywany w sposób od całkiem nieformalnego do bardzo formalnego
- główne cele: przedyskutowanie, podjęcie decyzji, ocena alternatyw, wyszukanie usterek, rozwiązanie problemów technicznych oraz sprawdzenie zgodności ze specyfikacją i standardami

Inspekcja

- prowadzona przez przeszkolonego moderatora (nie autora)
- zwykle sprawdzenie przez kolegów
- posiada zdefiniowane role
- posiada metryki
- posiada formalny proces oparty na regułach i listach kontrolnych
- posiada zdefiniowane kryteria wejścia i zakończenia
- przygotowanie przed spotkaniem przeglądowym
- raport z inspekcji zawierający listę uwag
- formalny proces kontroli wykonania napraw
    - opcjonalnie elementy doskonalenia procesów
- opcjonalnie wykorzystanie czytającego
- główny cel: wyszukanie usterek

Przejrzenia, przeglądy techniczne oraz inspekcje można wykonywać w grupie osób równych rangą - kolegów z tego samego szczebla organizacji. Taki przegląd nazywa się przeglądem koleżeńskim.

K2

3.2.4 Czynniki wpływające na powodzenie przeglądów

Następujące czynniki wpływają na sukces przeglądów:

- każdy przegląd ma jasno zdefiniowany cel
- w przegląd zaangażowani są ludzie odpowiedni do jego celu
- testerzy są wartościowymi przeglądającymi, którzy przyczyniają się do sukcesu przeglądu oraz poznają produkt, co pozwala im przygotować testy wcześniej
- znalezione usterki są przyjmowane pozytywnie i wyrażane w sposób obiektywny
- rozwiązuje się problemy personalne i psychologiczne (np. jest to pozytywnym doświadczeniem dla autora)
- przegląd jest prowadzony w atmosferze zaufania; wyniki przeglądu nie zostają użyte do oceny uczestników przeglądu
- stosuje się techniki przeglądania adekwatne do celów przeglądu, do typu i poziomu produktu oraz przeglądających
- jeżeli jest to potrzebne, zostają użyte listy kontrolne lub role do podniesienia skuteczności znajdowania usterek
- organizuje się szkolenia, szczególnie z bardziej formalnych technik takich jak inspekcja
- kierownictwo wspiera dobry proces przeglądu (np. przez przeznaczenie odpowiedniej ilości czasu w harmonogramach na zadania związane z przeglądem)
- kładzie się nacisk na uczenie się oraz doskonalenie procesów

3.3 Analiza statyczna przy pomocy narzędzi

K2 20min

Pojęcia
kompilator, złożoność, przepływ sterowania, przepływ danych, analiza statyczna

Celem analizy statycznej jest wyszukanie usterek w kodzie programu lub w modelach bez uruchamiania oprogramowania, które jest sprawdzane przez narzędzie używane w tym procesie. Miejscem, w którym kod jest wykonywany, są testy dynamiczne. Analiza statyczna może znaleźć usterki, które są trudne do znalezienia podczas testowania dynamicznego. Tak jak to było w przypadku przeglądów, analiza statyczna znajduje usterki, a nie awarie. Narzędzia do analizy statycznej analizują kod oprogramowania (np. przepływ sterowania lub przepływ danych) jak również wygenerowane wyjście w postaci HTMLa lub XMLa.

O wartości analizy statycznej stanowią następujące jej cechy:

- wczesne wykrywanie usterek, jeszcze przed wykonaniem testów

- wczesne wykrywanie podejrzanych aspektów kodu lub projektu przez wyliczenie miar, takich jak wysoki stopień złożoności
- identyfikacja defektów trudnych do wykrycia przez testowanie
- wykrywanie zależności i niespójności w modelach oprogramowania
- zwiększenie pielęgnowalności kodu i projektu
- zapobieganie defektom, jeżeli zastosowane zostają wnioski z analizy procesu rozwoju oprogramowania

Analiza statyczna zwykle wykrywa następujące typy usterek:

- odwołanie do niezainicjalizowanej zmiennej
- niespójne interfejsy pomiędzy modułami
- niewykorzystywane lub niepoprawnie zadeklarowane zmienne
- martwy kod
- brakująca albo błędna logika (pętle potencjalnie nieskończone)
- zbyt skomplikowane konstrukcje
- naruszenie standardów kodowania
- słabe punkty zabezpieczeń
- naruszenie reguł składni kodu i modeli oprogramowania

Narzędzia do analizy statycznej używane są zwykle przez programistów (do sprawdzania zgodności ze zdefiniowanymi regułami lub standardami kodowania) przed lub w trakcie testów modułowych lub integracyjnych lub też podczas wkładania kodu do narzędzia do kontroli wersji (commit). Mogą one też być wykorzystywane przez projektantów podczas modelowania oprogramowania. Narzędzia do analizy statycznej mogą zaraportować dużą liczbę ostrzeżeń, którymi trzeba dobrze zarządzać, żeby uzyskać jak największą skuteczność użycia narzędzia.

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

Literatura

3.2 IEEE 1028

3.2.2 Gilb,1993, van Veenendaal, 2004

3.2.4 Gilb,1993, IEEE 1028

3.3 van Veenendaal, 2004

4. Techniki projektowania testów

K3 285min

Cele nauczania dla technik projektowania testów.

Te cele określają co będziesz potrafił po zakończeniu modułu.

4.1 Proces rozwoju testów (K2)

LO-4.1.1 Kandydat odróżnia projekt testów od specyfikacji przypadków testowych oraz od procedury testowej. (K2)
LO-4.1.2 Kandydat potrafi porównać następujące pojęcia: warunek testowy, przypadek testowy, procedura testowa. (K2)
LO-4.1.3 Kandydat potrafi ocenić jakość przypadków testowych, przez sprawdzenie czy:

- mają jednoznaczne powiązania z wymaganiami
- zawierają wynik oczekiwany (K2)

LO-4.1.4 Kandydat potrafi utworzyć z przypadków testowych dobrze ustrukturalizowaną procedurę testową na poziomie szczegółowości odpowiadającym wiedzy testerów. (K3)

4.2 Kategorie technik projektowania testów (K2)

LO-4.2.1 Kandydat pamięta do czego przydatne są techniki projektowania testów oparte na specyfikacji (czarnoskrzynkowe) oraz techniki oparte na strukturze (białoskrzynkowe) oraz potrafi wymienić typowe dla nich techniki. (K1)
LO-4.2.2 Kandydat potrafi objaśnić cechy, podobieństwa oraz różnice pomiędzy technikami opartymi na specyfikacji, strukturze i doświadczeniu. (K2)

4.3 Techniki oparte na specyfikacji lub czarnoskrzynkowe (K3)

LO-4.3.1 Kandydat potrafi napisać przypadki testowe na podstawie podanych modeli oprogramowania używając techniki klas równoważności, analizy wartości brzegowych, testowania w oparciu o tablicę decyzyjną, testowania przejść pomiędzy stanami (K3)
LO-4.3.2 Kandydat potrafi wyjaśnić główny cel każdej z czterech technik testowania, na jakim poziomie testowania mogą zostać użyte i jak można dla nich zmierzyć pokrycie. (K2)
LO-4.3.3 Kandydat potrafi wyjaśnić na czym polega testowanie w oparciu o przypadki użycia i korzyści płynące z jego zastosowania. (K2)

4.4 Techniki oparte na strukturze lub białoskrzynkowe (K4)

LO-4.4.1 Kandydat potrafi opisać pojęcie pokrycia kodu i jego znaczenie. (K2)
LO-4.4.2 Kandydat potrafi wyjaśnić pojęcia: pokrycia instrukcji i pokrycia decyzji oraz podać powody, dla których te pojęcia mogą zostać użyte nie tylko na poziomie testów modułowych, ale na przykład też dla procedur biznesowych na poziomie testów systemowych). (K2)
LO-4.4.3 Kandydat potrafi napisać przypadki testowe dla danych przepływów sterowania stosując techniki testowania instrukcji i testowania decyzji. (K3)
LO-4.4.4 Kandydat potrafi ocenić kompletność testów według zdefiniowanych kryteriów zakończenia na podstawie pokrycia instrukcji i decyzji. (K4)

4.5 Techniki oparte na doświadczeniu (K2)

LO-4.5.1 Kandydat pamięta powody pisania przypadków testowych opierających się na intuicji, doświadczeniu oraz wiedzy na temat często spotykanych usterek. (K1)
LO-4.5.2 Kandydat potrafi porównać techniki oparte na doświadczeniu z technikami opartymi na specyfikacji. (K2)

4.6 Wybór technik testowania (K2)

LO-4.6.1 Kandydat potrafi podzielić techniki projektowania testów według ich dopasowania do danego kontekstu, podstawy testowania, modeli oraz cech oprogramowania. (K2)

4.1 Proces rozwoju testów

K2 15min

Pojęcia
specyfikacja przypadków testowych, projekt testu, harmonogram wykonania testu, specyfikacja procedury testowej, skrypt testowy, śledzenie

Proces rozwoju testów opisany w tym rozdziale może być wykonywany na wiele różnych sposobów, od bardzo nieformalnych bez dokumentacji lub z małą jej ilością do bardzo sformalizowanych (tak jak jest to opisane poniżej). Poziom sformalizowania zależy od kontekstu testowania, dojrzałości procesów rozwoju oprogramowania i testowania, ograniczeń czasowych, wymagań związanych z bezpieczeństwem i uregulowaniami prawnymi oraz zaangażowanych ludzi.

Podczas analizy testów, przeglądana jest dokumentacja podstawy testów w celu określenia co należy przetestować, czyli warunków testowych. Warunek testowy definiuje się jako element lub zdarzenie, które może zostać sprawdzone przez jeden lub więcej przypadków testowych (np. funkcja, transakcja, atrybut jakościowy lub element struktury).

Wdrożenie śledzenia powiązań warunków testowych ze specyfikacją i wymaganiami pozwala zarówno na wykonanie skutecznej analizy wpływu, gdy wymagania się zmieniają, jak i ustalenie pokrycia wymagań dla danego zbioru testów. Aby wybrać techniki projektowania testów, które zostaną użyte, podczas analizy stosowane jest szczegółowe podejście do testów oparte między innymi na analizie ryzyka (patrz rozdział 5).

Podczas projektowania testów tworzy się i opisuje przypadki testowe i dane testowe. Przypadek testowy składa się ze zbioru wartości wejściowych, warunków wstępnych, oczekiwanych wyników oraz warunków zakończenia utworzonych żeby pokryć pewne cele testów lub warunki testowe. "Standard for Software Test Documentation" (IEEE STD 829-1998) opisuje zawartość specyfikacji projektu testów (zawierającej warunki testowe) oraz specyfikacji przypadków testowych.

Jako część specyfikacji przypadku testowego powinny zostać określone wyniki oczekiwane. Powinny one zawierać opis wyjść, zmian w danych i stanie oprogramowania oraz inne skutki testu. Gdy wyniki oczekiwane nie są zdefiniowane, może się zdarzyć, że wiarygodne, ale błędne, wyniki zostaną uznane za poprawne. Najlepiej jest, gdy wyniki oczekiwane zostaną zdefiniowane przed wykonaniem testów.

Podczas implementacji testów przypadki testowe są rozwijane, implementowane, priorytetyzowane i układane w specyfikację procedur testowych (IEEE STD 829-1998). Procedura testowa zawiera kolejność działań podczas wykonania testu. Jeżeli testy są wykonywane przy pomocy narzędzia do wykonywania testów, ciąg akcji jest zawarty w skrypcie testowym (który stanowi automatyczną procedurę testową).

Różne procedury testowe i automatyczne skrypty testowe są następnie układane w harmonogram wykonania testów, który definiuje porządek wykonania procedur testowych i automatycznych skryptów testowych (o ile są obecne). Przy tworzeniu harmonogramu wykonania testów bierze się pod uwagę takie czynniki jak testy regresywne, priorytety oraz zależności techniczne i logiczne.

4.2 Kategorie technik projektowania testów

K2 15min

Pojęcia
czarnoskrzynkowa technika projektowania przypadków testowych, technika projektowania testów oparta na doświadczeniu, technika projektowania testów, białoskrzynkowa technika projektowania przypadków testowych

Celem technik projektowania testów jest zdefiniowanie warunków testowych, przypadków testowych i danych testowych.

Klasyczny podział wyróżnia techniki czarnoskrzynkowe oraz białoskrzynkowe. Czarnoskrzynkowe techniki projektowania testów (również nazywane technikami opartymi na specyfikacji) są sposobem na wywodzenie[1] oraz wybieranie warunków testowych, przypadków testowych i danych testowych bazującym na analizie podstawy testów. Można ich używać zarówno w testowaniu funkcjonalnym jak i niefunkcjonalnym. Techniki czarnoskrzynkowe z definicji nie wykorzystują żadnych informacji o strukturze wewnętrznej testowanego modułu lub systemu. Białoskrzynkowe techniki projektowania testów (również nazywane strukturalnymi lub technikami opartymi na strukturze) bazują na analizie struktury modułu lub systemu. Testy biało i czarnoskrzynkowe mogą również korzystać z doświadczenia programistów, testerów i użytkowników w celu określenia, co powinno zostać przetestowane.

Niektóre techniki da się przyporządkować jednoznacznie do jednej kategorii, inne zawierają elementy więcej niż jednej kategorii. W tym sylabusie techniki projektowania testów oparte na specyfikacji nazywane są technikami czarnoskrzynkowymi, a techniki oparte na strukturze – białoskrzynkowymi. Techniki oparte na doświadczeniu omówione są oddzielnie.

Cechy wspólne technik projektowania testów opartych na specyfikacji, to m.in.:

- w specyfikacji problemu do rozwiązania, oprogramowania lub jego komponentów używane są modele
- z tych modeli można, w sposób usystematyzowany, wywodzić przypadki testowe

Cechy wspólne technik projektowania testów opartych na strukturze, to m.in.:

- do tworzenia przypadków testowych wykorzystywana jest wiedza o tym jak oprogramowanie jest skonstruowane, np. kod źródłowy lub szczegółowy projekt
- można mierzyć stopień pokrycia istniejących przypadków testowych, można też w sposób usystematyzowany tworzyć nowe przypadki testowe w celu zwiększenia pokrycia

Cechy wspólne technik projektowania testów opartych na doświadczeniu, to m.in.:

- do tworzenia przypadków testowych wykorzystywane jest doświadczenie ludzi
  - wiedza testerów, programistów, użytkowników oraz innych interesariuszy o oprogramowaniu, jego środowisku i sposobie użytkowania
  - wiedza o prawdopodobnych defektach i ich położeniu

4.3 Techniki oparte na specyfikacji lub czarnoskrzynkowe

K3 150min

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

K3

4.3.1 Podział na klasy równoważności

W technice podziału na klasy równoważności wejścia programu lub systemu są dzielone na grupy, które powodują podobne zachowanie oprogramowania, więc wysoce prawdopodobne jest to, że są przetwarzane w ten sam sposób. Klasy równoważności można znaleźć dla danych poprawnych (wartości, które powinny zostać zaakceptowane) oraz niepoprawnych (wartości, które powinny zostać odrzucone). Klasy równoważności można znaleźć również dla wyjść, wartości wewnętrznych, wartości zależnych od czasu (np. przed lub po zajściu jakiegoś zdarzenia) oraz parametrów interfejsów (np. komponentów integrowanych i testowanych podczas testów integracyjnych). Testy można tak

zaprojektować, żeby pokrywały akceptowalne i nieakceptowalne klasy równoważności. Podział na klasy równoważności można zastosować na każdym poziomie testowania.

Technika podziału na klasy równoważności może zostać użyta do osiągnięcia celów pokrycia zarówno wejścia jak i wyjścia. Może zostać zastosowana do wartości wejściowych wprowadzanych ręcznie, przez interfejsy oraz do parametrów interfejsów podczas testów integracyjnych.

K3

4.3.2 Analiza wartości brzegowych

Istnieje większe prawdopodobieństwo, że oprogramowanie będzie się błędnie zachowywać dla wartości na krawędziach klas równoważności niż w ich środku, więc testowanie tych obszarów najprawdopodobniej wykryje błędy. Minimum i maksimum klasy równoważności to jej wartości brzegowe. Wartość brzegowa poprawnego przedziału jest nazywana poprawną wartością brzegową, a wartość brzegowa niepoprawnego przedziału – niepoprawną wartością brzegową. Testy można zaprojektować tak, żeby pokrywały zarówno poprawne jak i niepoprawne wartości brzegowe. Podczas projektowania testów tworzy się przypadek testowy dla każdej wartości brzegowej.

Analiza wartości brzegowych może zostać zastosowana na każdym poziomie testowania. Jest ona stosunkowo łatwa w użyciu i daje duże możliwości wykrywania usterek. Szczegółowa specyfikacja jest pomocna w znajdowaniu interesujących wartości brzegowych.

Technika ta jest często uważana za rozwinięcie techniki podziału na klasy równoważności lub innych technik czarnoskrzynkowych. Może być użyta dla klas równoważności wartości wprowadzanych przez interfejs użytkownika jak również, na przykład, dla przedziałów czasowych (np. time outów, wymagań na szybkość przetwarzania transakcji) lub zakresów tablic (np. rozmiar tablicy ma być 256x256).

K3

4.3.3 Testowanie w oparciu o tablicę decyzyjną

Tabele decyzyjne są dobrym sposobem na uchwycenie tych wymagań na system, które zawierają zależności logiczne, oraz na udokumentowanie wewnętrznej budowy systemu. Mogą być używane do zapisywania złożonych reguł biznesowych, które system ma obsługiwać. Podczas tworzenia tablic decyzyjnych analizuje się specyfikację systemu i wyszukuje warunki oraz wynikające z nich zachowanie systemu. Warunki wejściowe oraz zachowanie systemu często muszą być zapisane jako prawda lub fałsz (Boolean). Tabela decyzyjna zawiera warunki uruchamiające, często jako kombinacje prawdy i fałszu, dla wszystkich warunków wejściowych oraz działania wynikające z każdej z tych kombinacji. Każda kolumna tabeli odpowiada jednej regule biznesowej, która definiuje unikalną kombinację warunków i której skutkiem jest działanie związane z daną regułą biznesową. Standardowe pokrycie stosowane dla testowania w oparciu o tabelę decyzyjną wymaga zaprojektowania jednego testu dla każdej kolumny w tablicy, co zwykle oznacza wykorzystanie wszystkich kombinacji warunków uruchamiających.

Moc testowania w oparciu o tabelę decyzyjną leży w uwidocznieniu kombinacji warunków, które w innym przypadku mogłyby zostać pominięte w testach. Technika ta ma zastosowanie w każdej sytuacji, w której zachowanie oprogramowania zależy od kilku decyzji logicznych.

K3

4.3.4 Testowanie przejść między stanami

System może różnie odpowiadać w zależności od aktualnych warunków oraz od historii (od stanu). W takim przypadku zachowanie systemu można opisać diagramem przejść stanów (automatem skończonym). Pozwala to testerowi spojrzeć na oprogramowanie od strony stanów, przejść między stanami, wejść lub zdarzeń, które powodują zmiany stanów (przejścia) oraz na działania, które mogą wynikać z tych przejść. Stany testowanego systemu lub elementu są rozdzielne, zdefiniowane oraz ich liczba jest skończona. Tabela stanów pokazuje zależności pomiędzy stanami oraz wejściami i może uwypuklić przejścia nieprawidłowe. Testy można zaprojektować tak, żeby pokryły typowe sekwencje stanów, każdy stan, każde przejście, konkretny ciąg przejść lub żeby testowały przejścia nieprawidłowe.

Testowanie przejść między stanami jest często używane w testowaniu oprogramowania wbudowanego[2] oraz ogólnie w automatyce. Jednakże technikę tę można zastosować do zamodelowania obiektu biznesowego posiadającego określone stany lub do testowania przejść między formatkami (np. w aplikacjach internetowych lub scenariuszach biznesowych).

K2

4.3.5 Testowanie w oparciu o przypadki użycia

Testy można projektować na podstawie przypadków użycia. Przypadek użycia opisuje interakcje pomiędzy aktorami (użytkownikami lub systemami), które powodują powstanie wyniku wartościowego z punktu widzenia użytkownika lub klienta. Przypadek użycia może być opisany na wysokim poziomie abstrakcji (biznesowy przypadek użycia, poziom procesów biznesowych, niezawierający informacji o technologii) lub na poziomie systemowym (systemowy przypadek użycia na poziomie funkcjonalności systemu). Każdy przypadek użycia posiada warunki wstępne, które muszą zostać spełnione, żeby przypadek użycia został wykonany. Każdy przypadek użycia kończy się warunkami końcowymi. Są nimi widoczne rezultaty jego wykonania oraz stan systemu po zakończeniu przypadku użycia. Przypadki użycia zwykle posiadają scenariusz główny (tj. najbardziej prawdopodobny) oraz czasami scenariusze poboczne.

Przypadki użycia opisują „przepływy" procesu przez system bazując na najbardziej prawdopodobnym rzeczywistym jego użyciu, co sprawia że przypadki testowe wywiedzione z przypadków użycia najbardziej przydają się wykrywaniu usterek w przepływach procesów z rzeczywistego użytkowania systemu. Przypadki użycia bardzo przydają się w projektowaniu testów akceptacyjnych, w których ma brać udział klient/użytkownik. Pomagają również wykrywać defekty integracji spowodowane interakcją i interfejsami różnych modułów, co nie byłoby widoczne w testach modułowych. Projektowanie przypadków testowych z

przypadków użycia może zostać połączone z innymi technikami projektowania testów opartymi na specyfikacji.

4.4 Techniki oparte na strukturze lub białoskrzynkowe

K3 60min

Pojęcia
pokrycie kodu, pokrycie decyzji, pokrycie instrukcji, testowanie oparte na strukturze

Testowanie oparte na strukturze (białoskrzynkowe) bazuje na rozpoznanej strukturze oprogramowania lub systemu tak jak to widać w następujących przykładach:

- w testach modułowych: strukturą jest kod, to jest instrukcje, decyzje, rozgałęzienia lub nawet rozróżnialne ścieżki
- w testach integracyjnych: strukturą może być hierarchia wywołań (diagram, który pokazuje jak moduły wywołują inne moduły)
- w testach systemowych: strukturą może być budowa menu, proces biznesowy lub struktura strony webowej

W tym podrozdziale omówione są trzy strukturalne techniki projektowania testów związane pokryciem kodu bazujące na instrukcjach, decyzjach oraz rozgałęzieniach. W testowaniu decyzji można użyć diagramu przepływu sterowania, aby pokazać wyniki dla każdej decyzji.

K4

4.4.1 Testowanie i pokrycie instrukcji

W testowaniu instrukcji stosuje się pokrycie instrukcji, które polega na zmierzeniu, jaki odsetek instrukcji wykonywalnych został przetestowany przez zestaw testów. W technice tej projektuje się przypadki testowe, tak by wykonać określone instrukcje, zwykle po to żeby zwiększyć pokrycie.

Pokrycie instrukcji oblicza się przez podzielenie liczby wykonywalnych instrukcji pokrytych przez (zaprojektowane lub wykonane) przypadki testowe, przez liczbę wszystkich wykonywalnych instrukcji w testowanym kodzie.

K4

4.4.2 Testowanie i pokrycie decyzji

Pokrycie decyzji, spokrewnione z testowaniem gałęzi, polega na zmierzeniu jaki odsetek wyników decyzji (np. wyniku *prawda* lub *fałsz* instrukcji *if*) został przetestowany przez zestaw testów. W technice testowania decyzji projektuje się przypadki testowe tak aby pokryć określone wyniki decyzji. Gałęzie zaczynają się w punktach decyzyjnych i pokazują przekazanie sterowania do różnych miejsc w kodzie. Testowanie gałęzi różni się od testowania decyzji przez to, że skupia się na samych gałęziach.

Pokrycie decyzji jest wyliczane przez podzielenie liczby wyników decyzji pokrytych przez (zaprojektowane lub wykonane) przypadki testowe przez liczbę wszystkich wyników decyzji znajdujących się w testowanym kodzie.

Testowanie decyzji jest jedną z form testowania przepływu sterowania, ponieważ podąża za konkretnym przepływem sterowania przez punkty decyzyjne. Pokrycie decyzji jest mocniejsze niż pokrycie instrukcji. 100% pokrycia decyzji gwarantuje 100% pokrycia instrukcji, ale nie odwrotnie.

K1

4.4.3 Inne techniki oparte na strukturze

Istnieją techniki jeszcze mocniejsze niż pokrycie decyzji, na przykład pokrycie warunków lub wielokrotne pokrycie warunków.

Pojęcie pokrycia może zostać również zastosowane na innych poziomach testowania. Na przykład w testowaniu integracyjnym odsetek modułów, komponentów lub klas, które zostały przetestowane przez zestaw testów można wyrazić jako pokrycie modułów, komponentów lub klas.

W testowaniu strukturalnym przydatne jest wsparcie narzędziowe.

4.5 Techniki oparte na doświadczeniu

K2 30min

Pojęcia
testowanie eksploracyjne, atak (usterkowy)

Z testowaniem opartym na doświadczeniu mamy do czynienia, gdy testy projektuje się na podstawie wiedzy i intuicji testerów oraz ich doświadczenia z podobnymi aplikacjami i technologiami. Jeżeli zostanie ono użyte jako uzupełnienie technik systematycznych (zwłaszcza gdy zostanie zastosowane po nich), może okazać się użyteczne w uchwyceniu specjalnych przypadków testowych, których nie da się łatwo zaprojektować używając technik formalnych. Niemniej jednak jego skuteczność może okazać się bardzo różna w zależności od doświadczenia testerów.

Szeroko wykorzystywaną techniką opartą na doświadczeniu jest zgadywanie błędów. Ogólnie rzecz biorąc testerzy przewidują usterki bazując na swoim doświadczeniu. Ustrukturalizowane podejście do zgadywania błędów polega na sporządzeniu listy możliwych defektów i na zaprojektowaniu testów, które atakują te defekty. To systematyczne podejście jest nazywane atakiem usterkowym. Listy defektów i awarii można budować na podstawie doświadczenia, dostępnych danych na temat usterek i awarii oraz na ogólnej wiedzy dlaczego oprogramowanie nie działa.

Testowanie eksploracyjne polega na równoległym projektowaniu testów, ich wykonywaniu, zapisywaniu wyników oraz uczeniu się, bazując na zamówieniu na testy zawierającym cele testów i trzymając się ram czasowych. To podejście okazuje się najpożyteczniejsze, gdy nie

ma specyfikacji, albo jest ona niekompletna czy przestarzała, również w sytuacjach bardzo krótkiego czasu na testy. Przydaje się ono również do wzbogacenia i uzupełnienia bardziej formalnych testów. Może służyć też do kontroli procesu testowania w celu upewnienia się, że wszystkie najpoważniejsze usterki zostały wykryte.

4.6 Wybór technik testowania

K2 15min

Wybór, która technika testowania powinna zostać użyta, zależy od wielu czynników, m.in. typu systemu, regulacji prawnych, wymagań klienta lub kontraktu, poziomu ryzyka, typu ryzyka, celu testów, dostępnej dokumentacji, wiedzy testerów, czasu i budżetu, cyklu rozwoju oprogramowania, modelu przypadków użycia oraz doświadczenia związanego ze znajdowanymi typami usterek.

Niektóre z technik można z większą korzyścią zastosować tylko na niektórych poziomach testowania. Inne techniki można z równym powodzeniem stosować na wszystkich poziomach testów.

Podczas projektowania przypadków testowych testerzy zwykle stosują różne techniki projektowania testów włącznie z technikami opartymi na procesie, regułach oraz danych, po to żeby zapewnić odpowiednie pokrycie testowanego obiektu.

Literatura

4.1 Craig, 2002, Hetzel, 1998, IEEE STD 829-1998

4.2 Beizer, 1990, Copeland, 2004

4.3.1 Copeland, 2004, Myers, 1979

4.3.2 Copeland, 2004, Myers, 1979

4.3.3 Beizer, 1990, Copeland, 2004

4.3.4 Beizer, 1990, Copeland, 2004

4.3.5 Copeland, 2004

4.4.3 Beizer, 1990, Copeland, 2004

4.5 Kaner, 2002

4.6 Beizer, 1990, Copeland, 2004

Przypisy

1. derive
2. embedded software.

5. Zarządzanie testowaniem

K3 170 min

Cele nauczania dla zarządzania testami.

Te cele określają co będziesz potrafił po zakończeniu modułu.

5.1 Organizacja testów (K2)

LO-5.1.1 Kandydat uznaje ważność testowania niezależnego. (K1)
LO-5.1.2 Kandydat potrafi wyjaśnić korzyści i wady niezależnego testowania w organizacji. (K2)
LO-5.1.3 Kandydat uznaje potrzebę włączenia różnych członków zespołu podczas tworzenia zespołu testerskiego. (K1)
LO-5.1.4 Kandydat pamięta zadania typowego lidera testów oraz testera. (K1)

5.2 Planowanie i szacowanie testów (K2)

LO-5.2.1 Kandydat rozpoznaje różne poziomy i cele planowania testów. (K1)
LO-5.2.2 Kandydat potrafi omówić krótko cel i zawartość planu testów, projektu testów, procedury testowej zgodnie ze standardem dokumentacji testowania oprogramowania (Standard for Software Test Documentation IEEE STD 829-1998). (K2)
LO-5.2.3 Kandydat rozróżnia odmienne podejścia do testowania, takie jak analityczne, oparte na modelach, metodyczne, zgodne z procesem lub standardem, dynamiczne/heurystyczne, konsultatywne oraz regresywne. (K2)
LO-5.2.4 Kandydat odróżnia planowanie testów systemu od harmonogramowania ich wykonania. (K2)
LO-5.2.5 Kandydat potrafi napisać harmonogram wykonania testów dla danego zbioru przypadków testowych, uwzględniając ich priorytety oraz zależności logiczne i techniczne. (K3)
LO-5.2.6 Kandydat potrafi wymienić czynności przygotowania i wykonania testów, które należy wziąć pod uwagę przy planowaniu. (K1)
LO-5.2.7 Kandydat pamięta typowe czynniki, które mają wpływ na pracochłonność testowania. (K1)
LO-5.2.8 Kandydat odróżnia od siebie dwa koncepcyjnie odmienne podejścia do szacowania: podejścia bazujące na metrykach i podejścia wykorzystujące ekspertów. (K2)
LO-5.2.9 Kandydat potrafi rozpoznać i uzasadnić odpowiednie kryteria wejścia oraz zakończenia konkretnych poziomów testowania oraz grup przypadków testowych (np. testów integracyjnych, testów akceptacyjnych lub przypadków testowych w testach użyteczności). (K2)

5.3 Monitorowanie postępu testów i nadzór (K2)

LO-5.3.1 Kandydat potrafi wskazać najczęściej używane metryki do monitorowania przygotowania i wykonania testów. (K1)
LO-5.3.2 Kandydat potrafi wyjaśnić i porównać metryki stosowane w raportowaniu i kontroli testów (np. znalezione i poprawione usterki, zaliczone i niezaliczone testy). (K2)
LO-5.3.3 Kandydat potrafi omówić krótko cel i zawartość raportu podsumowującego testy zgodnie ze standardem dokumentacji testowania oprogramowania (IEEE STD 829-1998). (K2)

5.4 Zarządzanie konfiguracją (K2)

LO-5.4.1 Kandydat potrafi opisać krótko, w jaki sposób zarządzanie konfiguracją wspiera testowanie. (K2)

5.5 Ryzyka a testowanie (K2)

LO-5.5.1 Kandydat potrafi opisać ryzyko jako potencjalny problem, który zagrażałby osiągnięciu celów projektowych jednego lub kilku interesariuszy projektu. (K2)
LO-5.5.2 Kandydat pamięta, że poziom ryzyka jest określany przez prawdopodobieństwo wystąpienia oraz wpływ (potencjalną szkodę, jaką może uczynić gdy wystąpi). (K1)
LO-5.5.3 Kandydat rozróżnia elementy ryzyka projektowego i produktowego. (K2)
LO-5.5.4 Kandydat rozpoznaje typowe elementy ryzyka projektowego i produktowego. (K1)
LO-5.5.5 Kandydat potrafi opisać przy użyciu przykładów jak analiza ryzyka i zarządzanie ryzykiem mogą zostać wykorzystane przy planowaniu testów. (K2)

5.6 Zarządzanie incydentami (K3)

LO-5.6.1 Kandydat zna zawartość raportu incydentu zgodnie ze standardem dokumentacji testowania oprogramowania (IEEE STD 829-1998). (K1)
LO-5.6.2 Kandydat potrafi napisać raport incydentu zawierający opis awarii zaobserwowanej podczas testowania. (K3)

5.1 Organizacja testów

K2 30min

Pojęcia
tester, lider testów, kierownik testów

5.1.1 Organizacja testów a ich niezależność

Skuteczność wykrywania usterek w testach i przeglądach może zostać podniesiona przez zaangażowanie niezależnych testerów. Niezależność może występować w różnych wariantach, włączając w to następujące:

- brak niezależnych testerów, programiści testują swój własny kod
- niezależni testerzy wewnątrz zespołu projektowego
- niezależny zespół testowy lub grupa testerów wewnątrz organizacji podlegająca kierownikowi projektu lub zarządowi
- niezależni testerzy z departamentów biznesowych lub społeczności użytkowników
- niezależni specjaliści od określonych typów testów takich jak użyteczności, zabezpieczeń lub certyfikacji oprogramowania (którzy przeprowadzają certyfikację oprogramowania na zgodność z regulacjami prawnymi lub standardami)
- niezależni testerzy, którzy zostali wynajęci lub są na zewnątrz organizacji

W projektach dużych, złożonych lub przy wytwarzaniu aplikacji krytycznych ze względu na bezpieczeństwo, zwykle najlepiej mieć kilka poziomów testów, z których niektóre lub wszystkie wykonywane są przez niezależnych testerów. Programiści mogą uczestniczyć w testach, szczególnie na niższych poziomach, ale ich brak obiektywności często ogranicza skuteczność. Niezależni testerzy mogą zostać upoważnieni do zdefiniowania i egzekwowania procesu testowania i ról z nim związanych, ale powinni to robić tylko w przypadku posiadania zdecydowanego poparcia ze strony kierownictwa.

Korzyści z niezależności:

- niezależni testerzy widzą inne i odmienne usterki niż twórcy oraz nie mają uprzedzeń
- niezależny tester może zweryfikować założenia poczynione podczas specyfikacji i implementacji systemu

Wady niezależności:

- izolacja od zespołu deweloperskiego (jeżeli niezależność jest całkowita)
- programiści mogą utracić poczucie odpowiedzialności za jakość
- niezależni testerzy mogą być postrzegani jako wąskie gardło lub obwiniani za opóźnienia w wydaniach

Zadania związane z testowaniem mogą być wykonywane przez ludzi pełniących konkretne role testerskie lub przez ludzi pełniących inne role np. kierownika projektu, menedżera

jakości, programistę, eksperta biznesowego lub dziedzinowego, ludzi związanych z serwisem operacyjnym lub IT.

K2

5.1.2 Zadania lidera testów oraz testera

W tym sylabusie zostały omówione dwie role testerskie: lider testów oraz tester. Zadania wykonywane przez ludzi pełniących te dwie role zależą od kontekstu projektowego i produktowego, pełniących te role oraz organizacji.

Czasami lider testów nazywany jest kierownikiem testów lub koordynatorem testów. Rolę lidera testów może pełnić kierownik projektu, kierownik zespołu wytwórczego, kierownik zapewnienia jakości lub kierownik zespołu testowego. W większych projektach mogą być obecne dwa stanowiska: lidera oraz kierownika testów. Zwykle to lider testów planuje, monitoruje i kontroluje zadania testowe i czynności tak jak to było podane w podrozdziale 1.4.

Typowe zadania lidera testów to:

- koordynowanie strategii oraz planu testów z kierownikami projektu i innymi interesariuszami
- tworzenie lub przeglądanie strategii testów w projekcie oraz polityki testowania w organizacji
- przedstawianie perspektywy testowania w innych zadaniach projektowych takich jak planowanie integracji
- planowanie testów, uwzględniając ich kontekst oraz rozumiejąc cele testów i ryzyko, włącznie z wyborem podejścia do testów, szacowaniem czasu, pracochłonności i kosztów testowania, zdobywaniem zasobów, definiowaniem poziomów testów, cykli testowych oraz planowaniem zarządzania incydentami
- inicjowanie specyfikacji, przygotowania, implementacji i wykonania testów, monitorowanie ich wyników oraz sprawdzanie spełnienia kryteriów zakończenia
- zmiana planów z uwzględnieniem wyników oraz postępu testów (czasami udokumentowanego w raporcie statusu testów), a także podejmowanie działań koniecznych do rozwiązania problemów
- zorganizowanie odpowiedniego zarządzania konfiguracją testaliów w celu zwiększenia możliwości śledzenia powiązań
- wprowadzenie odpowiednich metryk do mierzenia postępu testów i oceny jakości testowania oraz produktu
- decydowanie co powinno zostać zautomatyzowane, w jakim stopniu i w jaki sposób
- wybór narzędzi wspierających testy oraz organizacja dla testerów szkoleń z użycia tych narzędzi
- decydowanie o implementacji środowiska testowego
- sporządzanie raportów podsumowujących testy bazując na informacjach zebranych podczas testów

Typowe zadania testera to:

- przeglądanie i wnoszenie wkładu do planów testów
- analiza, przegląd oraz ocena wymagań użytkownika, specyfikacji oraz modeli ze szczególnym uwzględnieniem testowalności
- tworzenie specyfikacji testów
- współtworzenie środowiska testowego (często jest to koordynacja pracy administratorów oraz osób odpowiedzialnych za sieć)
- przygotowanie i pozyskanie danych testowych
- implementacja testów na wszystkich poziomach, wykonanie i logowanie testów, ocena wyników oraz dokumentowanie odchyleń od oczekiwanych wyników
- używanie narzędzi do administracji lub zarządzania testami oraz narzędzi do monitorowania testów, gdy jest to wymagane
- automatyzacja testów (może być wspierana przez programistę lub eksperta od automatyzacji testów)
- pomiar wydajności modułów i systemów (o ile ma zastosowanie)
- przeglądanie testów wytworzonych przez innych

Ludzie, którzy pracują nad analizą testów, projektowaniem testów, konkretnymi typami testów lub ich automatyzacją mogą być specjalistami w swoich rolach. W zależności od poziomu testów oraz ryzyka produktowego i projektowego, różne osoby mogą pełnić rolę testera, będąc do pewnego stopnia niezależnymi. Na poziomach modułowym i integracyjnym testerami zwykle są programiści, na poziomie testów akceptacyjnych - eksperci biznesowi oraz użytkownicy, a testerami w produkcyjnych testach akceptacyjnych - operatorzy.

5.2 Planowanie i szacowanie testów

K3 40min

Pojęcia
podejście do testów, strategia testów

K2

5.2.1 Planowanie testów

Podrozdział ten wyjaśnia cel planowania testów w projektach deweloperskich i wdrożeniowych oraz czynności pielęgnacyjnych. Planowanie może zostać udokumentowane w głównym planie testów lub w oddzielnych planach testów dla każdego poziomu np. dla testów systemowych oraz testów akceptacyjnych. Wzorce dokumentów planistycznych dla testów zawarte są w dokumencie "Standard for Software Test Documentation" (IEEE STD 829-1998).

Na planowanie testów wpływ ma polityka testowania w organizacji, zakres testów, cele, ryzyko, ograniczenia, krytyczność, testowalność oraz dostępność zasobów. Im dłużej trwa planowanie projektu i testów, tym więcej informacji jest dostępnych i tym więcej szczegółów może być włączone do planowania.

Planowanie testów jest czynnością stałą i wykonuje się je dla wszystkich procesów i zadań cyklu życia oprogramowania. Informacje zwrotne z czynności testowych są używane do wykrywania zmieniających się ryzyk, tak że planowanie może zostać dopasowane do bieżącej sytuacji w projekcie.

K2

5.2.2 Czynności związane z planowaniem testów

W planowanie testów dla całego systemu lub jego części mogą wchodzić następujące czynności:

- ustalenie zakresu i ryzyka oraz zidentyfikowanie celów testowania
- zdefiniowanie ogólnego podejścia do testowania włącznie z definicją poziomów testów oraz kryteriów wejścia i zakończenia
- integrowanie i koordynowanie zadań testowych z innymi zadaniami cyklu życia oprogramowania: zakupami, dostawami, rozwojem, działaniem produkcyjnym oraz pielęgnacją
- podejmowanie decyzji co testować, którym rolom będą przypisane które zadania testowe, jak zadania testowe powinny być wykonane oraz jak powinno się oceniać wyniki testów
- harmonogramowanie analizy i projektowania testów
- harmonogramowanie implementacji, wykonania i oceny testów
- przydzielanie zasobów do zadań testowych
- definiowanie ilości, poziomu szczegółowości, struktury oraz wzorców dokumentacji testowej
- wybór metryk do monitorowania i kontroli przygotowania i wykonania testów, naprawy defektów oraz elementów ryzyka
- decydowanie o poziomie szczegółowości procedur testowych, aby dostarczyć wystarczającą ilość informacji dla powtarzalnego przygotowania i wykonania testów

K2

5.2.3 Kryteria wejścia

Kryteria wejścia definiują warunki pozwalające na rozpoczęcie testów na początku poziomu testów lub, gdy zbiór testów jest gotowy do wykonania.

Kryteria wejścia zwykle mogą zawierać następujące zagadnienia:

- dostępność i gotowość środowiska testowego
- gotowość narzędzi testowych w środowisku testowym
- dostępność testowalnego kodu
- dostępność danych testowych

5.2.4 Kryteria zakończenia

K2

Celem kryteriów zakończenia jest zdefiniowanie momentu zakończenia testów na danym poziomie testów lub gdy zbiór testów osiągnął określony cel.

Typowo kryteria zakończenia mogą składać się z:

- miar staranności, takich jak pokrycie kodu, funkcjonalności lub ryzyka
- estymat gęstości błędów lub miar niezawodności
- kosztu
- istniejącego ryzyka, takiego jak niepoprawione usterki lub brak pokrycia pewnych obszarów
- harmonogramów np. zdefiniowanych na podstawie czasu do wypuszczenia produktu na rynek.

K2

5.2.5 Szacowanie testów

Istnieją dwa podejścia do szacowania pracochłonności testów:

- podejście oparte na metrykach

  szacowanie pracochłonności testów bazując na pomiarach minionych lub podobnych projektów lub bazujące na typowych wartościach

- podejście oparte na ekspertach

  szacowanie zadań przez ich przyszłych wykonawców lub przez ekspertów

Gdy pracochłonność zostanie już oszacowana można przyporządkowywać zasoby do zadań oraz szkicować harmonogram.

Pracochłonność testów może zależeć od wielu czynników, a w tym:

- cech produktu:

  jakości specyfikacji oraz innych informacji używanych w modelach testowych (tj. w podstawie testów), wielkości produktu, złożoności dziedziny problemu, wymagań na niezawodność oraz zabezpieczenie oraz wymagań na dokumentację

- cech procesu produkcyjnego:

  stabilności organizacji, użytych narzędzi, procesu testowego, umiejętności ludzi oraz presji czasu

- wyników testów:

liczby usterek oraz pracochłonności napraw i przeróbek

K2

5.2.6 Podejście do testowania, strategia testowania

Podejście do testów jest implementacją strategii testów w konkretnym projekcie. Podejście do testów jest definiowane i uszczegóławiane w planach testów oraz projektach testów. Zwykle zawiera decyzje podejmowane na podstawie celów projektu (testowego) oraz oceny ryzyka. Stanowi ono punkt wyjściowy do planowania procesu testowania, wyboru technik projektowania testów i stosowanych typów testów, a także do definiowania kryteriów wejścia i zakończenia.

Wybrane podejście zależy od kontekstu i może uwzględniać ryzyka, niebezpieczeństwa oraz wymagane bezpieczeństwo, dostępne zasoby oraz umiejętności, technologię, naturę systemu (np. system niestandardowy vs. z półki), cele testów i uregulowania prawne.

Typowe podejścia do testów to:

- podejścia analityczne, takie jak testy oparte na ryzyku, w którym testowanie jest kierowane na obszary o największym ryzyku
- podejścia oparte na modelach, takie jak testowanie stochastyczne wykorzystujące informacje statystyczne na temat współczynników awarii (takich jak modele wzrostu niezawodności oprogramowania) lub wykorzystania oprogramowania (takich jak profile operacyjne)
- podejścia metodyczne, takie jak podejścia oparte na awariach (włącznie ze zgadywaniem błędów i atakami usterkowymi), oparte na doświadczeniu, na listach kontrolnych lub na atrybutach jakościowych
- podejścia zgodne ze standardem lub procesem, takie jak te określone przez standardy przemysłowe lub metodyki zwinne
- podejścia dynamiczne i heurystyczne, takie jak testowanie eksploracyjne, w którym testowanie bardziej reaguje na zdarzenia podczas testów niż jest wykonywane według planu i w którym wykonywanie testów i ocena wyników dzieją się równolegle
- podejścia konsultatywne, w których pokrycie testowe jest sterowane głównie przez wskazówki i porady ekspertów technologicznych lub biznesowych z zewnątrz zespołu testowego
- podejścia regresywne, w których używa się powtórnie istniejących materiałów testowych, rozbudowanej automatyzacji regresywnych testów funkcjonalnych oraz standardowych zestawów testów

Różne podejścia mogą zostać połączone, np. w dynamiczne testy oparte na ryzyku.

5.3 Monitorowanie postępu testów i nadzór

K2 20min

Pojęcia
gęstość błędów, współczynnik awarii, nadzorowanie testów, monitorowanie testów, sumaryczny raport z testów

K1

5.3.1 Monitorowanie postępu testów

Celem monitorowania jest uzyskanie informacji zwrotnych oraz uzyskanie wglądu w przebieg zadań testowych. Informacje, które mają być monitorowane, mogą zostać zebrane ręcznie lub automatycznie i mogą zostać użyte do pomiarów spełnienia kryteriów zakończenia (np. pokrycia). Metryki mogą również zostać wykorzystane do oceny postępów według zaplanowanego harmonogramu i budżetu.

Często wykorzystywanymi metrykami są:

- procent pracy wykonanej przy przygotowywaniu przypadków testowych lub odsetek przygotowanych przypadków testowych
- procent prac wykonanych przy przygotowywaniu środowiska testowego
- wykonanie przypadków testowych (np. liczba wykonanych/nie wykonanych przypadków testowych oraz liczba przypadków testowych zaliczonych/niezaliczonych
- informacje o usterkach (np. gęstość błędów, defekty znalezione i poprawione, współczynnik awarii oraz wyniki testów)
- pokrycie testami wymagań, ryzyka i kodu
- subiektywne zaufanie testerów do produktu
- daty kamieni milowych
- koszt testowania, włączając w to porównanie kosztu do korzyści dla znalezienia kolejnego defektu lub wykonania kolejnego testu

K2

5.3.2 Raportowanie testów

Raportowanie testów podaje podsumowanie projektu testowego, a w tym:

- co się zdarzyło w czasie testowania, np. daty spełnienia kryteriów zakończenia
- analizę informacji oraz metryk wspierającą rekomendację oraz decyzje co do przyszłych działań, takich jak pozostałe usterki, opłacalność dalszego testowania, pozostałe obszary ryzyka oraz poziom zaufania do testowanego oprogramowania

Zarys raportu podsumowującego testy przedstawiony jest w dokumencie „Standard for Software Documentation” (IEEE STD 829-1998).

Pomiary powinny być wykonywane w trakcie oraz na koniec poziomu testów, żeby ocenić:

- dopasowanie celów testów do poziomu testów
- adekwatność wybranego podejścia do testów
- skuteczność testów w odniesieniu do celów testowania

K2

5.3.3 Kierowanie testami

Kierowanie testami to wszystkie działania zarządcze lub korekcyjne podjęte na skutek zebranych i zaraportowanych informacji i pomiarów. Działania te mogą dotyczyć dowolnego zadania testowego lub mogą wpływać na dowolną czynność lub zadanie związane z cyklem życia oprogramowania.

Przykładami działań związanych z kierowaniem testami są:

- podejmowanie decyzji na podstawie informacji uzyskanych z monitorowania testów
- zmiana priorytetów testów, kiedy zmaterializuje się ryzyko (np. oprogramowanie zostanie dostarczone z opóźnieniem)
- zmiana harmonogramu testów związana z dostępnością lub niedostępnością środowiska testowego
- ustanowienie kryteriów wejścia wymagających, aby programista zretestował poprawki (wykonał testy potwierdzające) zanim włączy je do wydania.

5.4 Zarządzanie konfiguracją

K2 10min

Pojęcia
zarządzanie konfiguracją, kontrola wersji

Celem zarządzania konfiguracją jest ustanowienie i utrzymanie integralności produktów (modułów, danych i dokumentacji) związanych z oprogramowaniem lub systemem przez cały projekt lub cykl życia produktu.

Z punktu widzenia testowania, zarządzanie konfiguracją może wymagać zapewnienia, że:

- wszystkie elementy oprogramowania zostały zidentyfikowane, poddane kontroli wersji, są śledzone, powiązane między sobą oraz z elementami deweloperskimi (przedmiotem testów), tak żeby można utrzymać możliwość śledzenia zmian i powiązań[6] przez cały proces testowy
- odwołania w dokumentacji testowej do wszystkich zidentyfikowanych dokumentów oraz elementów softwarowych są jednoznaczne

Zarządzanie konfiguracją pomaga testerom jednoznacznie wskazać (i odtworzyć) testowany element, dokumenty testowe, testy oraz jarzmo testowe.

Podczas planowania testów powinny zostać wybrane, udokumentowane i wdrożone procedury zarządzania konfiguracją oraz infrastruktura (narzędzia).

5.5 Ryzyko a testowanie

K2 30 min

Pojęcia
ryzyko produktowe, ryzyko projektowe, ryzyko, testowanie oparte na ryzyku

Ryzyko może zostać zdefiniowane, jako możliwość wystąpienia zdarzenia, niebezpieczeństwa, zagrożenia lub jakiejś sytuacji powodującej niepożądane konsekwencje lub potencjalny problem. Poziom ryzyka jest określony przez prawdopodobieństwo wystąpienia niekorzystnego zdarzenia oraz jego wpływ (szkoda, jaką może wyrządzić to zdarzenie).

K2

5.5.1 Obszary ryzyka projektowego

Ryzyko projektowe, to obszary ryzyka otaczające zdolność projektu do osiągnięcia postawionych przed nim celów, takie jak:

- czynniki organizacyjne
  - braki w umiejętnościach, szkoleniach lub personelu
  - problemy kadrowe
  - problemy polityczne takie jak:
    - problemy z testerami komunikującymi swoje potrzeby oraz wyniki testów
    - brak reakcji zespołu w związku z informacjami pozyskanymi podczas testów i przeglądów (np. brak doskonalenia procesów produkcji i testowania)
  - nieprawidłowe nastawienie i oczekiwania od testowania (np. niedocenianie wartości znajdowania błędów podczas testowania)
- problemy techniczne
  - problemy ze zdefiniowaniem poprawnych wymagań
  - stopień, w jakim wymagania mogą zostać spełnione przy istniejących ograniczeniach
  - środowisko testowe niegotowe na czas
  - spóźniona konwersja danych, planowanie migracji oraz rozwój i testowanie narzędzi do migracji i konwersji danych
  - niska jakość projektu, kodu, danych konfiguracyjnych, danych testowych i testów
- problemy z dostawcami
  - niewywiązywanie się dostawców ze zobowiązań
  - problemy z kontraktami

Podczas analizowania, zarządzania i łagodzenia powyższych obszarów ryzyka kierownik testów postępuje według uznanych zasad kierowania projektami. Wzorzec planu testów ze standardu „Standard for Software Documentation” (IEEE STD 829-1998) wymaga wymienienia elementów ryzyka oraz zabezpieczeń przed nimi.

5.5.2 Obszary ryzyka produktowego

Potencjalne obszary wystąpienia awarii (przyszłych niekorzystnych zdarzeń lub niebezpieczeństw) w oprogramowaniu lub systemie nazywane są ryzykiem produktowym, ponieważ stanowią ryzyko dla jakości produktu. Mogą to być: K2

- dostarczanie awaryjnego oprogramowania
- możliwość wyrządzenia szkody człowiekowi lub firmie przez oprogramowanie lub sprzęt
- niedostateczne atrybuty oprogramowania (np. funkcjonalność, niezawodność, użyteczność lub wydajność)
- niska jakość lub brak spójności danych (np. problemy z migracją danych, problemy z konwersją danych, problemy z przekazywaniem danych, naruszenie standardów danych)
- oprogramowanie, które nie spełnia założonych funkcji

Ryzyka używa się do określenia, kiedy rozpocząć testowanie oraz co powinno zostać dokładniej przetestowane. Testowanie jest wykorzystywane do zredukowania ryzyka wystąpienia niekorzystnych zdarzeń lub do zredukowania ich wpływu.

Obszary ryzyka produktowego należą do specjalnego rodzaju ryzyk projektowych. Testowanie, jako działanie kontrolujące ryzyko, dostarcza informacji na temat istniejących obszarów ryzyka przez pomiar skuteczności usuwania krytycznych usterek oraz przez planów awaryjnych.

Oparte na ryzyku podejście do testowania umożliwia w sposób proaktywny obniżanie ryzyka produktowego rozpoczynając już od wstępnej fazy projektu. Zawiera ono identyfikację obszarów ryzyka produktowego, co umożliwia użycie ich jako wskazówek w planowaniu i kontroli testów, specyfikacji, przygotowaniu oraz wykonaniu testów. W podejściu do testów opartym na ryzyku zidentyfikowane obszary ryzyka mogą zostać wykorzystane do:

- określenia technik testowania, które powinny zostać użyte
- określenia zakresu testów
- priorytetyzacji testów w celu znalezienia usterek krytycznych tak wcześnie jak to tylko możliwe
- określenia, czy nie można użyć działań niezwiązanych z testowaniem w celu redukcji ryzyka (np. przeszkolenie niedoświadczonych projektantów)

Testowanie oparte na ryzyku bazuje na grupowej wiedzy i obserwacjach interesariuszy projektu w celu określenia obszarów ryzyka oraz poziomów testowania wymaganych, żeby odnieść się do nich.

W celu zapewnienia minimalizacji szansy wystąpienia awarii produktu, zarządzanie ryzykiem daje zdyscyplinowane podejście do:

- oceny (powtarzanej regularnie), co może pójść nie tak (ryzyka)
- ustalenia, którymi obszarami ryzyka należy się zająć
- wdrożenia działań zarządzających tymi obszarami ryzyka

Dodatkowo testy mogą wspierać identyfikację nowych obszarów ryzyka, mogą pomóc w ustaleniu, które obszary ryzyk powinny zostać zminimalizowane oraz mogą zmniejszyć niepewność co do ryzyka.

5.6 Zarządzanie incydentami

K3 40min

Pojęcia
rejestracja incydentu, zarządzanie incydentami, raport o incydencie

Ponieważ jednym z celów testowania jest znajdowanie usterek, rozbieżności pomiędzy oczekiwanymi i rzeczywistymi wynikami muszą być zapisywane jako incydenty. Incydenty powinny być analizowane i może się okazać, że stanowią defekty. Odpowiednie czynności, mówiące jak należy postępować z incydentami i defektami, powinny zostać zdefiniowane. Incydenty i defekty powinny być śledzone od momentu wykrycia i klasyfikacji do naprawy i potwierdzenia rozwiązania. Żeby być w stanie zarządzać wszystkimi incydentami aż do ich zamknięcia, organizacja powinna wprowadzić proces zarządzania incydentami oraz reguły klasyfikacji.

Incydenty można zgłaszać podczas programowania, przeglądów, testowania lub użytkowania produktu softwarowego. Mogą być zgłoszone dla problemów w kodzie lub działającym systemie, a także dla dokumentów dowolnego typu, czyli wymagań, dokumentów deweloperskich, dokumentów testowych lub informacji dla użytkownika takich jak pomoc lub instrukcja instalacji.

Raporty incydentów istnieją w celu:

- dostarczenia programistom i innym stronom informacji na temat problemów, aby umożliwić identyfikację, wyodrębnienie i naprawę, jeżeli okaże się to konieczne
- dostarczenia liderom testów środków do śledzenia jakości testowanego systemu oraz postępu testów
- dostarczenia pomysłów na doskonalenie procesu testowania

Raport incydentów może zawierać następujące szczegóły:

- datę zgłoszenia, zgłaszającą organizację oraz autora
- wyniki oczekiwane oraz rzeczywiste
- wskazanie na element testowy (element konfiguracji) oraz na środowisko
- proces cyklu życia oprogramowania lub systemu w którym incydent został zaobserwowany
- opis incydentu, w celu umożliwienia odtworzenia i rozwiązania, włącznie z logami, zrzutami baz danych oraz zrzutami ekranu
- obszar lub stopień wpływu na interesy interesariuszy
- stopień wpływu na system
- pilność, priorytet naprawy

- status incydentu (np. otwarty, odłożony, duplikat, oczekujący na naprawę, naprawiony i oczekujący na retest, zamknięty)
- podsumowania, rekomendacje oraz zgody
- zagadnienia globalne, takie jak inne obszary, na które mogą mieć wpływ zmiany związane z incydentem
- historia zmian, np. ciąg czynności podjętych przez członków zespołu w celu wyizolowania incydentu, jego naprawy oraz potwierdzenia tej naprawy
- referencje, włączając w to identyfikator specyfikacji przypadku testowego, który wykrył problem

Struktura raportu incydentu jest również przedstawiona w dokumencie „Standard for Software Test Documentation” (IEEE STD 829-1998).

Literatura

5.1.1 Black, 2001, Hetzel, 1998

5.1.2 Black, 2001, Hetzel, 1998

5.2.5 Craig, 2002, IEEE STD 829-1998, Kaner 2002

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

5.4 Craig, 2002

5.5.2 Black, 2001, IEEE STD 829-1998

5.6 Black, 2001, IEEE STD 829-1998

.

6. Testowanie wspierane narzędziami

80 min

Cele nauczania dla testowania wspieranego narzędziami.

Te cele określają co będziesz potrafił po zakończeniu modułu.

6.1 Typy narzędzi testowych (K2)

LO-6.1.1 Kandydat potrafi podzielić różne typy narzędzi testowych według ich celów, według czynności w podstawowym procesie testowym oraz w cyklu życia oprogramowania. (K2)
LO-6.1.3 Kandydat potrafi wyjaśnić pojęcie "narzędzie testowe" oraz cel wsparcia narzędziowego dla testów. (K2)

6.2 Skuteczne użycie narzędzi, potencjalne korzyści i ryzyko (K2)

LO-6.2.1 Kandydat potrafi opisać krótko potencjalne korzyści oraz ryzyko automatyzacji testów oraz wspierania testów narzędziami. (K2)
LO-6.2.2 Kandydat pamięta specjalne uwarunkowania dla narzędzi wspierających wykonywanie testów, analizę statyczną oraz zarządzanie testami. (K1)

6.3 Wdrażanie narzędzi w organizacji (K1)

LO-6.3.1 Kandydat potrafi wymienić główne zasady wprowadzania narzędzi do organizacji. (K1)
LO-6.3.2 Kandydat potrafi wymienić cele dowodu słuszności pomysłu (proof-of-concept) w ocenie narzędzia oraz cele fazy pilotażowej we wdrażaniu tego narzędzia. (K1)
LO-6.3.3 Kandydat uznaje, że poza samym zakupem narzędzia potrzebne jest też dobre wsparcie w jego użyciu. (K1)

6.1 Typy narzędzi testowych

K2 45min

Pojęcia
narzędzie do zarządzania konfiguracją, narzędzie mierzące pokrycie, narzędzie do debagowania, narzędzie do analizy dynamicznej, narzędzie do zarządzania incydentami, narzędzie do testów obciążeniowych, narzędzie do modelowania, narzędzie monitorujące, narzędzie do testów wydajnościowych, efekt próbnika, narzędzie do zarządzania wymaganiami, narzędzie wspomagające przeglądy, narzędzie do zabezpieczeń, narzędzie do analizy statycznej, narzędzie do testowania przeciążającego, komparator testowy, narzędzie do przygotowywania danych testowych, narzędzie do projektowania testów, jarzmo testowe, narzędzie do wykonywania testów, narzędzie do zarządzania testami, struktura do testów jednostkowych

K2

6.1.1 Znaczenie i cel wsparcia narzędziowego dla testów

Narzędzia testowe mogą być wykorzystywane w jednej lub wielu czynnościach wspierających testowanie. Mogą to być:

1. narzędzia używane wprost w testach takie jak narzędzia wspierające wykonanie testów, narzędzia generujące dane testowe i narzędzia porównujące wyniki
2. narzędzia wspomagające zarządzanie procesem testowym takie jak narzędzia do zarządzania testami, wynikami testów, danymi, wymaganiami, incydentami, defektami itd. oraz narzędzia raportujące i monitorujące wykonanie testów
3. narzędzia używane w rozpoznaniu (eksploracji), np. narzędzia monitorujące dostęp do plików przez aplikację
4. dowolne narzędzie wspierające testy (w takim znaczeniu arkusz kalkulacyjny jest również narzędziem testowym)

W zależności od kontekstu wsparcie narzędziowe dla testów może mieć jeden lub kilka z poniższych celów:

- zwiększenie efektywności czynności testowych przez zautomatyzowanie powtarzających się zadań lub wsparcie dla czynności testowych wykonywanych ręcznie takich jak planowanie testów, projektowanie testów, raportowanie i monitorowanie testów
- automatyzacja czynności, które wymagałyby wielkich nakładów, gdyby były wykonywane ręcznie (np. testy statyczne)
- automatyzacja czynności, które nie mogą być wykonane ręcznie (np. testy aplikacji klient-serwer na wielką skalę)
- zwiększenie niezawodności testów (np. przez automatyzację porównywania dużej ilości danych lub symulacje)

Termin "struktura testowa"[1] jest również często używany w przynajmniej trzech znaczeniach:

- reużywalne i rozszerzalne biblioteki testowe, które mogą zostać wykorzystane do zbudowania narzędzi testowych (zwane również jarzmami testowymi[2])
- typ projektu automatyzacji testów (np. testowanie sterowane danymi, testowanie oparte na słowach kluczowych)
- ogólny proces wykonania testów

W tym sylabusie pojęcie "struktura testowa" jest używane w pierwszych dwóch znaczeniach, tak jak to zostało opisane w podrozdziale 6.1.6.

K2

6.1.2 Klasyfikacja narzędzi testowych

Istnieje wiele narzędzi wspierających różne aspekty testowania. Narzędzia testowe można podzielić na wiele sposobów: według celów, na komercyjne, darmowe, o otwartym kodzie, shareware, według wykorzystywanej technologii i tak dalej. W tym sylabusie narzędzia są podzielone na grupy według wspieranych przez nie czynności testowych.

Niektóre narzędzia wspierają jeden rodzaj czynności. Inne mogą być przydatne w różnych czynnościach testowych, ale zostały zaklasyfikowane do tej grupy, z którą są najmocniej związane. Narzędzia od jednego dostawcy, a szczególnie narzędzia, które zostały zaprojektowane do współdziałania, mogą zostać powiązane w jeden pakiet.

Niektóre typy narzędzi są inwazyjne w tym sensie, że samo narzędzie może wpływać na wynik rzeczywisty testu. Na przykład czasy uzyskane podczas testów wydajnościowych mogą się różnić, ponieważ narzędzie wykonuje dodatkowe instrukcje, albo można uzyskać różne wyniki pokrycia kodu. Zjawisko to jest nazywane efektem próbnika.

Niektóre z narzędzi testowych są przeznaczone bardziej dla programistów (np. w testach modułowych lub testach integracji modułów). Narzędzia te zostały oznaczone literką D na poniższej liście.

6.1.3 Wsparcie narzędziowe dla zarządzania testowaniem i testami

Narzędzia do zarządzania przydają się we wszystkich czynnościach testowych przez cały cykl życia oprogramowania. K1

Narzędzia do zarządzania testami

Narzędzia te dostarczają interfejsów do wykonywania zadań, śledzenia defektów oraz zarządzania wymaganiami, jak również wspierają analizę ilościową i raportowanie na temat obiektów testów. Wspierają również śledzenie powiązań pomiędzy obiektami testów i mogą posiadać własne mechanizmy zarządzania wersjami lub interfejs do zewnętrznych narzędzi zarządzających wersjami.

Narzędzia do zarządzania wymaganiami

Narzędzia te przechowują tekst wymagań, ich atrybuty (np. priorytet), generują unikalne identyfikatory i wspierają śledzenie powiązań pomiędzy wymaganiami, a poszczególnymi testami. Mogą one również pomóc w znalezieniu niespójnych lub brakujących wymagań.

Narzędzia do zarządzania incydentami

Narzędzia te przechowują i zarządzają raportami incydentów, to jest defektów, awarii, żądań zmian lub zauważonych problemów i anomalii. Pomagają też zarządzać cyklem życia incydentów, opcjonalnie mogą wspomagać analizy statystyczne.

Narzędzia do zarządzania konfiguracją

Chociaż nie są one w ścisłym tego słowa znaczeniu narzędziami testowymi, są konieczne do przechowywania i zarządzania wersjami testaliów i innego oprogramowania, w szczególności gdy konfigurowane jest więcej niż jedno środowisko sprzętowo-programowe (wersje systemów operacyjnych, kompilatory, przeglądarki itp.).

K1

6.1.4 Wsparcie narzędziowe dla testów statycznych

Narzędzia wspierające testy statyczne stanowią opłacalny sposób na znajdowanie większej ilości defektów we wcześniejszych fazach życia oprogramowania.

Narzędzia wspierające przeglądy

Narzędzia te wspomagają proces przeglądu, listy kontrolne, wytyczne dla przeglądów i są wykorzystywane do przechowywania i komunikowania komentarzy z przeglądów, raportów defektów oraz pracochłonności. Mogą również wspierać przeglądy on-line, co jest przydatne, gdy zespół jest duży lub rozproszony geograficznie.

Narzędzia do analizy statycznej

Narzędzia te pomagają programistom i testerom jeszcze przed testami dynamicznymi D

przez wymuszanie standardów kodowania (włącznie z bezpiecznym programowaniem), analizę struktur i zależności. Mogą również pomóc w planowaniu i analizie ryzyka przez dostarczanie metryk kodu (np. złożoności).

Narzędzia do modelowania

Narzędzia te używane są w walidacji modeli oprogramowania (np. fizycznego modelu danych w bazach relacyjnych) do wymieniania niezgodności i wyszukiwania defektów. Narzędzia te pomagają też często w generowaniu przypadków testowych z modeli. D

K1

6.1.5 Wsparcie narzędziowe dla specyfikacji testów

Narzędzia do projektowania testów

Narzędzia te wykorzystywane są do generowania wejść do testów, wykonywalnych testów, lub wyroczni testowych z wymagań, graficznych interfejsów użytkownika, modeli projektowych (stanów, danych lub obiektów) lub kodu.

Narzędzia do przygotowywania danych testowych

Narzędzia do przygotowywania danych testowych przetwarzają bazy danych, pliki lub transmisję danych by uzyskać dane do wykorzystania podczas wykonywania testów. Korzyścią z ich stosowania jest to, że dane produkcyjne przeniesione na środowisko testowe są dla ochrony anonimizowane.

K1

6.1.6 Wsparcie narzędziowe dla wykonania testów oraz logowania

Narzędzia do wykonania testów

Narzędzia do wykonywania testów umożliwiają automatyczne lub półautomatyczne wykonywanie testów z wykorzystaniem zapisanych wejść i oczekiwanych wyników, poprzez użycie języka skryptowego, a także zwykle tworzą log (dziennik) testowy dla każdego przebiegu testów. Mogą one zostać również użyte do nagrania testów, zwykle też wykorzystują język skryptowy lub konfigurację opartą na GUI do parametryzowania danych lub dostosowywania testów na inne sposoby.

Jarzma testowe/struktury do testów jednostkowych

Jarzmo testów modułowych[3] lub struktura testowa[4] wspomaga testy komponentów lub części systemu przez symulację środowiska, w którym element testowy będzie działał, dostarczając makiety (sterowników testowych i zaślepek). D

Komparatory testowe

Komparatory znajdują różnice pomiędzy plikami, bazami danych oraz wynikami testów. Narzędzia wspomagające wykonywanie testów zwykle zawierają komparatory dynamiczne, ale porównania po wykonaniu testów mogą być dokonywane przez odrębne narzędzia. Komparator testowy może wykorzystać wyrocznię testową szczególnie w przypadkach gdy jest zautomatyzowany.

Narzędzia mierzące pokrycie

Narzędzia te, w sposób inwazyjny lub nieinwazyjny, mierzą odsetek określonych struktur kodu (instrukcji, gałęzi lub decyzji, modułów lub wywołań funkcji), które zostały pokryte przez zbiór testów. D

Narzędzia do testowania zabezpieczeń

Narzędzia te oceniają cechy oprogramowania związane z zabezpieczeniem. Wchodzi w to ocena zdolności oprogramowania do ochrony poufności danych, spójności, uwierzytelniania, autoryzacji, dostępności oraz niezaprzeczalności. Narzędzia do testowania zabezpieczeń są zwykle związane z konkretną technologią, platformą i celem.

K1

6.1.7 Wsparcie narzędziowe dla wydajności i monitorowania

Narzędzia do analizy dynamicznej

Narzędzia do analizy dynamicznej znajdują usterki, które ujawniają się jedynie podczas

działania oprogramowania, takie jak zależności czasowe lub wycieki pamięci. Narzędzia tego typu są zwykle używane podczas testów modułowych lub testów integracji modułów oraz podczas testów warstw pośrednich[5]. D

Narzędzia do testów wydajnościowych/narzędzia do testów obciążeniowych/narzędzia do testów przeciążeniowych

Narzędzia do testów wydajnościowych monitorują oraz raportują zachowanie systemu w różnych zasymulowanych warunkach użytkowania uwzględniających liczbę równolegle pracujących użytkowników, ich wzorzec aktywacji[6], częstość i względny procent transakcji. Symulacja obciążenia jest uzyskiwana przez utworzenie wirtualnych użytkowników wykonujących wybrany zbiór transakcji, rozproszonych na wiele maszyn testowych, które są powszechnie nazywane generatorami obciążenia.

Narzędzia do monitorowania

Narzędzia do monitorowania bezustannie analizują, weryfikują i raportują wykorzystanie konkretnych zasobów systemowych i ostrzegają o możliwych problemach w obsłudze żądań.

6.1.8 Wsparcie narzędziowe dla różnych obszarów zastosowań

Ocena jakości danych K1

Dane stanowią centrum wielu projektów, takich jak projekty konwersji/migracji danych, czy aplikacje takie jak hurtownie danych, a ich krytyczność i wielkość może się różnić. W takich okolicznościach muszą zostać użyte narzędzia do oceny jakości danych do przejrzenia i sprawdzenia konwersji danych oraz reguł migracji, po to żeby zapewnić, że przetworzone dane są poprawne, kompletne i zgodne ze zdefiniowanymi standardami.

Istnieją też inne narzędzia wspierające testy użyteczności.

6.2 Skuteczne użycie narzędzi, potencjalne korzyści i ryzyko

K2 20min

Pojęcia
testowanie sterowane danymi, testowanie oparte o słowa kluczowe, język skryptowy

K2

6.2.1 Potencjalne korzyści i ryzyko wsparcia narzędziowego dla testów (dla wszystkich narzędzi)

Sam zakup narzędzia lub wzięcie go w leasing nie gwarantuje jeszcze sukcesu. Każdy typ narzędzia może wymagać wykonania dodatkowego wysiłku do osiągnięcia prawdziwych i trwałych korzyści. Z używaniem każdego narzędzia wiążą się potencjalne korzyści i możliwości, ale wiąże się też z tym pewne ryzyko.

Potencjalnymi korzyściami z używania narzędzi są:

- zredukowana zostaje powtarzająca się praca (np. uruchamianie testów regresywnych, powtórne wprowadzanie tych samych danych testowych oraz sprawdzanie zgodności ze standardami kodowania)
- zwiększa się spójność i powtarzalność (np. testy wykonywane przez narzędzie w tej samej kolejności i z tą samą częstością oraz testy wywiedzione z wymagań)
- ocena jest obiektywna (np. miary statyczne, pokrycie)
- łatwiejszy jest dostęp do danych o testach i testowaniu (np. statystyki i wykresy obrazujące postęp testów, współczynniki występowania incydentów oraz wydajność)

Ryzyko związane z narzędziami do testowania obejmuje:

- nierealistyczne oczekiwania od narzędzia (co do funkcjonalności lub łatwości użycia)
- niedoszacowanie czasu, kosztów oraz pracochłonności wstępnego wdrożenia narzędzia (włączając w to szkolenia oraz ekspertów zewnętrznych)
- niedoszacowanie czasu i pracochłonności potrzebnych do osiągnięcia znaczących i trwałych korzyści z narzędzia (włączając w to potrzebę wykonania zmian w procesie testowym oraz ciągłe doskonalenie sposobu wykorzystania narzędzia)
- niedoszacowanie pracochłonności utrzymania artefaktów testowych wygenerowanych przez narzędzie

- zbytnie poleganie na narzędziu (zastąpienie narzędziem projektowania testów lub wykorzystywanie narzędzia, gdy testowanie ręczne byłoby lepsze)
- niewykorzystywanie kontroli wersji testaliów w narzędziu
- lekceważenie zależności i problemów z współpracą krytycznych narzędzi takich jak, narzędzia do zarządzania wymaganiami, narzędzia do kontroli wersji, narzędzia do zarządzania incydentami, narzędzia do śledzenia defektów oraz narzędzia od różnych dostawców
- słaba reakcja dostawcy w ramach wsparcia, nowych wersji oraz poprawiania usterek
- ryzyko wstrzymania projektu dla narzędzia darmowego / open-source
- nieprzewidziane, takie jak niezdolność do wspierania nowej platformy

K1

6.2.2 Specjalne uwagi dla niektórych typów narzędzi

Narzędzia do wykonywania testów

Narzędzia do wykonywania testów uruchamiają skrypty zaprojektowane tak, aby zaimplementować testy przechowywane elektronicznie. Ten typ narzędzi często wymaga wykonania znaczącej ilości pracy, zanim zostaną osiągnięte istotne korzyści.

Nagrywanie akcji wykonywanych przez testy ręczne wydaje się być atrakcyjne, ale nie skaluje się do dużej liczby zautomatyzowanych skryptów testowych. Nagrany skrypt jest liniową reprezentacją z konkretnymi danymi i akcjami będącymi częścią każdego skryptu. Taki typ skryptu może okazać się niestabilny, gdy wystąpią niespodziewane zdarzenia.

Testowanie sterowane danymi wydziela wejścia testowe (dane testowe), zwykle do arkusza kalkulacyjnego, i używa bardziej ogólnego skryptu testowego, który może odczytać dane wejściowe i wykonać ten sam skrypt dla różnych danych. Testerzy, którzy nie znają języków skryptowych, mogą wtedy tworzyć dane testowe dla tych predefiniowanych skryptów.

Istnieją też inne techniki wykorzystywane w testowaniu sterowanym danymi, w których zamiast stałych danych zapisanych w arkuszu kalkulacyjnym, dane są generowane przy użyciu algorytmów sterowanych parametrami konfigurowalnymi w czasie ich działania i podanymi aplikacji. Na przykład, aby uzyskać powtarzalność, narzędzie może używać algorytmu, generującego przypadkowe identyfikatory użytkowników, który jest inicjowany stałą wartością.

W podejściu sterowanym słowami kluczowymi arkusz kalkulacyjny zawiera słowa kluczowe opisujące akcje do wykonania (nazywane również słowami akcji) oraz dane testowe. Testerzy (nawet jeżeli nie znają języków skryptowych) mogą w takim przypadku definiować testy przy użyciu słów kluczowych, które zostają dostosowane do testowanej aplikacji.

Techniczna znajomość języków skryptowych jest konieczna przy każdym z wyżej wymienionych podejść (albo przez testerów, albo przez specjalistów od automatyzacji testów).

Niezależnie od tego którakolwiek z technik skryptowych zostanie użyta oczekiwane wyniki dla każdego testu muszą być przechowywane do późniejszych porównań.

Narzędzia do analizy statycznej

Gdy narzędzia do analizy statycznej zostają użyte na kodzie źródłowym mogą wymusić stosowanie się do standardów kodowania, ale takie narzędzie zostanie użyte na kodzie już istniejącym może wygenerować dużą liczbę komunikatów. Ostrzeżenia nie powodują zatrzymania kompilacji kodu, ale powinny być zaadresowane, tak żeby utrzymanie kodu było w przyszłości łatwiejsze. Najwłaściwszym podejściem do wdrażania tych narzędzi jest wyłączenie niektórych komunikatów na początku, a następnie stopniowe ich włączanie.

Narzędzia do zarządzania testami

Narzędzia do zarządzania testami muszą się komunikować z innymi narzędziami lub arkuszami kalkulacyjnymi w celu dostarczenia użytecznej informacji w formacie, który najbardziej pasuje do potrzeb organizacji.

6.3 Wdrażanie narzędzi w organizacji

K1 15min

Pojęcia
-

Główne aspekty do wzięcia pod uwagę podczas wyboru narzędzia dla organizacji to:

- ocena dojrzałości organizacji, mocnych i słabych stron oraz identyfikacja możliwości doskonalenia procesu testowania wspieranego narzędziami
- ocena według jasnych wymagań oraz obiektywnych kryteriów
- wykonanie dowodu słuszności pomysłu (proof-of-concept) z użyciem narzędzia testowego, po to żeby zbadać, czy jest ono skuteczne dla danego testowanego oprogramowania w ramach istniejącej infrastruktury lub po to, żeby określić jakie zmiany w infrastrukturze są potrzebne do skutecznego użycia narzędzia
- ocena dostawcy (włącznie ze szkoleniami, wsparciem oraz aspektami komercyjnymi) lub firm udzielających wsparcia w przypadku narzędzi niekomercyjnych
- identyfikacja wymagań wewnętrznych na doradztwo i szkolenia w użyciu narzędzia
- ocena potrzeb szkoleniowych z uwzględnieniem obecnych umiejętności automatyzacji testów przez zespół testowy
- szacowanie stosunku korzyści do kosztów na podstawie konkretnego przypadku biznesowego

Wdrażanie wybranego narzędzia w organizacji zaczyna się od projektu pilotażowego, który ma następujące cele:

- szczegółowe zapoznanie się z narzędziem
- ocena czy i jak narzędzie pasuje do obowiązujących procesów i praktyk oraz ustalenie, co ewentualnie należałoby zmienić

- ustalenie standardów użycia, zarządzania, przechowywania oraz pielęgnacji narzędzia oraz artefaktów testowych (np. wypracowanie konwencji nazewnictwa plików i testów, stworzenie bibliotek oraz zdefiniowanie modularyzacji zestawów testów)
- ocena, czy korzyści zostaną osiągnięte przy rozsądnych kosztach

Czynnikami wpływającymi na sukces wdrożenia narzędzia w organizacji są:

- stopniowe wdrażanie narzędzia w pozostałej części organizacji
- adaptacja i udoskonalenie procesu tak, aby pasował do sposobu używania narzędzia
- zapewnienie szkoleń oraz doradztwa nowym użytkownikom
- zdefiniowanie wytycznych co do użycia narzędzia
- wdrożenie sposobu na zbieranie użytecznych informacji z wykorzystania narzędzia
- monitorowanie wykorzystania narzędzia oraz osiąganych korzyści
- zapewnienie wsparcia dla zespołu testowego w użyciu danego narzędzia
- zbieranie wniosków z wykorzystania narzędzia przez wszystkie zespoły

Literatura

6.2.2 Buwalda, 2001, Fewster, 1999

6.3 Fewster, 1999

Przypisy

1. test framework
2. test harness
3. unit test harness
4. framework
5. middleware
6. ramp-up pattern
7. proof of concept.

7. Literatura

Normy

ISTQB Glossary of terms used in Software Testing Version 2.1

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

[IEEE 829- 1998] IEEE Std 829™ (1998/2005) IEEE Standard for Software Test Documentation; zob. rozdz. 2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6

[IEEE 1028] IEEE Std 1028™ (2008) IEEE Standard for Software Reviews and Audits; zob. rozdz.3.2

[IEEE 12207] IEEE 12207/ISO/IEC 12207-2008, Software life cycle processes; zob rozdz. 2.1

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

Książki

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

[Black, 2001] Black, R. (2001) *Managing the Testing Process* (3rd edition), John Wiley & Sons: New York; zob. rozdz. 1.1, 1.2, 1.4, 1.5, 2.3, 2.4, 5.1, 5.2, 5.3, 5.5, 5.6

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

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

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

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

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

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

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

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

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

8. Załącznik A – Pochodzenie planu

Historia tego dokumentu

Dokument został przygotowany w latach 2004 -2011 przez grupę roboczą składającą się z członków wyłonionych przez International Software Testing Qualifications Board (ISTQB®). Początkowo był przeglądany przez wybrany panel przeglądowy a potem przez reprezentantów wybranych z międzynarodowej społeczności zajmującej się testowaniem oprogramowania. Zasady zastosowane przy tworzeniu tego dokumentu pokazane są w Załączniku C.

Dokument ten jest planem dla Międzynarodowego Certyfikatu Podstawowego w Testowaniu Oprogramowania, pierwszego poziomu międzynarodowej kwalifikacji zaaprobowanego przez ISTQB (www.istqb.org).

Cele Certyfikatu Podstawowego

- Poznanie testowania jako podstawowej i profesjonalnej specjalizacji inżynierii oprogramowania.
- Zapewnienie standardu dla rozwoju kariery testera.
- Umożliwienie rozpoznawania profesjonalnie wykwalifikowanych testerów przez pracodawców, klientów i kolegów oraz podwyższenie statusu testerów.
- Promowanie konsekwentnych i dobrych praktyk testowania w dyscyplinach inżynierii oprogramowania.
- Zidentyfikowanie tematów testowania, które są istotne i wartościowe dla przemysłu.
- Umożliwienie dostawcom oprogramowania wynajęcie certyfikowanych testerów i dzięki temu uzyskanie handlowej przewagi nad ich konkurentami przez zareklamowanie własnej polityki zatrudnienia testerów.
- Zapewnienie testerom i osobom zainteresowanym testowaniem okazji nabycia rozpoznawanych na arenie międzynarodowej kwalifikacji w tym temacie.

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

- Umożliwienie porównania umiejętności testowania w różnych krajach.
- Umożliwienie testerom łatwiejszego przekraczania granic krajów.
- Umożliwienie wielonarodowym/międzynarodowym projektom wspólnego zrozumienia zagadnień z zakresu testowania.
- Zwiększenie liczby wykwalifikowanych testerów na świecie.
- Posiadanie większego wpływu/wartości jako inicjatywa umocowana międzynarodowo niż specyficzne podejście z jednego kraju.
- Rozwinięcie wspólnej międzynarodowej grupy zrozumienia i wiedzy o testowaniu dzięki planowi i terminologii oraz wzrost poziomu wiedzy na temat testowania u wszystkich uczestników.
- Promowanie testowania jako zawodu w wielu krajach.
- Umożliwienie testerom uzyskania rozpoznawanych kwalifikacji w ich ojczystych językach.
- Umożliwienie dzielenia się wiedzą i zasobami pomiędzy krajami.
- Zapewnienie międzynarodowego poznania testerów i kwalifikacji z powodu uczestnictwa z wielu krajów.

Wymagania wobec kandydatów

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

Kandydaci powinni mieć co najmniej sześciomiesięczne doświadczenie w wytwarzaniu oprogramowania albo w testach akceptacyjnych lub systemowych.

Kandydaci powinni wziąć udział w kursie ISTQB (akredytowanym przez narodową organizację uznaną przez ISTQB).

Okoliczności powstania i historia Podstawowego Certyfikatu w Testowaniu Oprogramowania

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

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

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

9. Załącznik B – Cel nauki i poziomy wiedzy

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

Poziom K1: Zapamiętać (K1)

Pojęcia: zapamiętać, powtórzyć, rozpoznać, wiedzieć.

Kandydat rozpoznaje, pamięta i umie określić termin lub pojęcie.

Przykład

Rozpoznanie definicji „awarii": jako:

- „brak funkcjonalności użytkownika końcowego lub innego interesariusza"

lub jako

- „niezgodność rzeczywistego działania modułu lub systemu z działaniem lub wynikiem oczekiwanym".

Poziom K2: Zrozumieć (K2)

Pojęcia: podsumowywać, klasyfikować, porównywać, przypisywać, przeciwstawiać, podawać przykłady, interpretować, tłumaczyć, reprezentować, dedukować, wnioskować, kategoryzować

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

Przykłady

Wyjaśnij, dlaczego testy powinny być tworzone najwcześniej, jak to możliwe:

- By odnaleźć defekty, w czasie, gdy ich usunięcie jest tańsze.
- By znaleźć najważniejsze defekty w pierwszej kolejności.

Wyjaśnić podobieństwa i różnice pomiędzy testowaniem integracyjnym i systemowym:

- Podobieństwa testowanie więcej niż jednego modułu i możliwość testowania aspektów niefunkcjonalnych.
- Różnice: testy integracyjne koncentrują się na interfejsach i wzajemnych oddziaływaniach, a testy systemowe skupiają się na aspektach działania systemu jako całości, pełnych procesach biznesowych („end-to-end").

Poziom K3: Zastosować (K3)

Pojęcia: zastosować, wykonać, użyć, postępować zgodnie z procedurą, zastosować procedurę

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

Przykład

Kandydat potrafi

- zidentyfikować wartości graniczne dla poprawnych/ważnych i niepoprawnych/nieważnych danych podzielonych na klasy równoważności.
- wybrać przypadki testowe z podanego diagramu przejść stanów, aby pokryć wszystkie przejścia.

Poziom K4: Analizować (K4)

Pojęcia: analizować, rozróżniać, wybierać, kategoryzować, koncentrować się, przypisywać cechę, rozkładać na części, oceniać, osądzać, monitorować, koordynować, tworzyć, syntetyzować, generować, tworzyć hipotezy, planować, projektować, budować, implementować.

Kandydat powinien umieć podzielić informacje związane z procedurą lub techniką testowania na części składowe w celu lepszego ich zrozumienia oraz odróżnić fakty od wniosków. Typowe zastosowanie to analiza dokumentu, analiza oprogramowania, analiza sytuacji w projekcie i propozycja akcji mających na celu rozwiązanie problemu lub zakończenie zadania.

Referencje

(Dla poznawczych poziomów celów nauki)

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

10. Załącznik C – Zasady dotyczące Planu poziomu podstawowego ISTQB / SJSI

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

10.1 Ogólne zasady

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

10.2 Bieżąca treść

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

10.3 Cele nauki

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

10.4 Ogólna struktura

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

Referencje

SR1. Źródła i referencje zostaną podane dla koncepcji w konspekcie, aby pomóc dostawcom szkoleń znaleźć więcej informacji na wybrany temat. (REFERENCJE)
SR2. Tam, gdzie nie ma zidentyfikowanych i jasnych źródeł powinno się dostarczyć więcej szczegółów w konspekcie. Na przykład, definicje są w słowniku, a więc w konspekcie przytoczono jedynie terminologie. (SZCZEGÓŁY BEZ REFERENCJI)

Źródła informacji

Terminologia użyta w konspekcie, dla pojęć używanych w testowaniu oprogramowania, zdefiniowana jest w Słowniku ISTQB®. Wersja słownika jest dostępna na stronie ISTQB® lub SJSI.
Lista zalecanych książek na temat testowania oprogramowania podana jest w konspekcie.
Główna lista książek jest częścią sekcji Literatura.

11. Załącznik D – Informacja dla dostawców szkoleń

Dla każdego z głównych tematów planu przeznaczony jest określony w minutach czas jego trwania. Celem tego jest zarówno wyznaczenie względnych proporcji poszczególnych sekcji tematycznych akredytowanego szkolenia względem siebie, jak i wyznaczenie przybliżonego minimalnego czasu, który należy przeznaczyć na nauczanie poszczególnych tematów.

Dostawcy szkoleń mogą poświęcić im więcej czasu, zaś kandydaci mogą spędzić dodatkowy czas na lekturze i własnych badaniach. Kolejność omawiania zagadnień podczas kursu nie musi być taka sama jak w konspekcie.

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

Wszystkie obszary tematyczne planu z poziomu K3 i K4 wymagają ćwiczeń praktycznych dołączonych do materiałów szkoleniowych.

12. Załącznik E – Uwagi do wydania.

1. Zmiany w celach uczenia (LO) zawierają objaśnienia

    a. Zmiana sformułowań w następujących LO (zawartość oraz poziom pozostał niezmieniony): LO-1.2.2, LO-1.3.1, LO-1.4.1, LO-1.5.1, LO-2.1.1, LO-2.1.3, LO-2.4.2, LO-4.1.3, LO-4.2.1, LO-4.2.2, LO-4.3.1, LO-4.3.2, LO-4.3.3, LO-4.4.1, LO-4.4.2, LO-4.4.3, LO-4.6.1, LO-5.1.2, LO-5.2.2, LO-5.3.2, LO-5.3.3, LO-5.5.2, LO-5.6.1, LO-6.1.1, LO-6.2.2, LO-6.3.2.
    b. LO-1.1.5 został przeformułowany i podniesiony do K2, ponieważ można spodziewać się porównywania pojęć związanych z defektami.
    c. LO-1.2.3 (K2) został dodany. Odpowiadająca mu treść istniała już w sylabusie w wersji 2007.
    d. LO-3.1.3 (K2) zawiera teraz LO-3.1.3 oraz LO-3.1.4
    e. LO–3.1.4 został usunięty z sylabusa 2010, ponieważ jest częściowo nadmierny w porównaniu z LO-3.1.3
    f. LO-3.2.1 został przeformułowany w celu utrzymania spójności z zawartością sylabusa 2010
    g. LO-3.3.2 został zmieniony i jego poziom został zmieniony z K1 na K2 dla spójności z LO-3.1.2
    h. LO-4.4.4 został ujednoznaczniony i został przeniesiony z K3 do K4. Powód: LO-4.4.4 już było sformułowane na poziomie K4
    i. LO-6.1.2 (K1) został usunięty z sylabusa 2010 i zastąpiony LO-6.1.3 (K2)

2. Spójne użycie terminu "podejście do testów" zgodne z definicją w słowniku. Termin strategia testów nie będzie już wymagany.
3. Podrozdział 1.4 zawiera teraz pojęcie śledzenia powiązań pomiędzy podstawą testów i przypadkami testowymi.
4. Rozdział 2.x zawiera teraz przedmiot testów oraz podstawę testów.
5. Retesty są teraz głównym terminem, tak jak w słowniku, a nie testowanie potwierdzające.
6. Aspekt jakości danych został dodany w kilku miejscach sylabusa: jakość danych i ryzyko w rozdziałach 2.2, 5.5, 6.1.8.
7. Został dodany podrozdział "5.2.3 Kryteria wejścia". Powód: Spójność z kryteriami zakończenia (kryteria wejścia zostały dodane do LO-5.2.9).
8. Użycie terminów strategia testów oraz podejście do testów zgodne z ich definicjami w słowniku.
9. Rozdział 6.1 został skrócony, gdyż opisy narzędzi były za długie w stosunku do 45 minut przeznaczonych na ten materiał
10. Został wydany standard IEEE 829:2008. Ta wersja sylabusa nie uwzględnia nowego wydania standardu. Sekcja 5.2 odwołuje się do dokumentu Główny Plan Testów. Zawartość Głównego Planu Testów jest pokazana według koncepcji że dokument Plan Testów może dotyczyć różnych poziomów planowania. Można utworzyć plany testów dla poszczególnych poziomów oraz plan testów dla całego projektu. Ten ostatni jest nazywany w sylabusie i w słowniku Głównym Planem Testów.
11. Kodeks etyczny został przeniesiony z poziomu zaawansowanego (CTAL) na podstawowy (CTFL).

Zmiany wprowadzone w wydaniu korekcyjnym (maintenance release) 2011:

1. Ogólne: Working Party zastąpione przez Working Group
2. Ogólne: ISTQB zastąpione przez ISTQB®tam gdzie to jest potrzebne
3. Ponieważ cele uczenia są opisane w dodatku B, tylko ogólny opis pozostawiono we wstępie do sylabusa
4. Sekcja 1.6: Został usunięty poziom nauczania, gdyż postanowiono nie definiować LO dla kodeksu etycznego
5. Sekcje 2.2.1, 2.2.2, 2.2.3 oraz 2.2.4, 3.2.3: Poprawiono formatowanie list
6. Sekcja 2.2.2: Słowo awaria było niepoprawne w tekście "…który moduł lub system jest przyczyną danej awarii" i zostało zastąpione słowem defekt.
7. Sekcja 2.3: Formatowanie listy celów testowania w odniesieniu do typów testów
8. Sekcja 2.3.4: Debagowanie opisane zgodnie z definicją w słowniku ISTQB w wersji 2.1
9. Sekcja 3.2.1: Ponieważ czynności przeglądu formalnego zostały źle sformatowane proces miał 12 głównych czynności zamiast 6. Niniejsza poprawka powoduje, że lista ta jest zgodna z sylabusem CTFL 2007 oraz sylabusem CTAL 2007
10. Sekcja 4.3.5: zmiana tekstu "pomiędzy aktorami, włączając w to użytkowników i system" na "pomiędzy aktorami (użytkownikami i systemami)
11. Sekcja 4.3.5: alternatywna ścieżka zamieniona na alternatywny scenariusz
12. Sekcja 4.4.2: Dodano zdanie uściślające przedmiot zainteresowań testowania gałęzi w celu wyjaśnienia terminu testowanie gałęzi
13. Sekcja 6.1: Nagłówek "6.1.1 Zrozumienie znaczenia i celu narzędziowego wsparcia dla testów (K2) zamienione na "6.1.1 Znaczenie i cel wsparcia narzędziowego dla testów"
14. Sekcja 7 (Książki): Trzecie wydanie [Black, 2001] zastępuje drugie wydanie
15. Dodatek E: LO zmienione pomiędzy wersjami 2007 i 2010 są teraz poprawnie wymienione.

Uwaga: Kilka zmian podanych w angielskiej wersji sylabusa nie zostały tu podane, ponieważ dotyczą zmian i literówek w języku angielskim.

13. Indeks