Certyfikowany Tester
Syllabus dla Poziomu Zaawansowanego

Wersja 2007

Stowarzyszenie Jakości Systemów Informatycznych

International Software Testing Qualifications Board

Advanced Level Working Party : Bernard Homes (przewodniczący), Graham Bath, Rex Black, Sigrid Eldh, Jayapradeep Jiothis, Paul Jorgensen, Vipul Kocher, Judy McKay, Klaus Olsen, Randy Rice, Jürgen Richter, Eric Riou Du Cosquer, Mike Smith, Geoff Thompson, Erik Van Veenendaal; 2006-2007

Tłumaczenie z języka angielskiego oraz udział w przeglądach: Joanna Gajewska (Nowakowska) (przewodnicząca), Karolina Zmitrowicz, Radosław Smilgin, Michał Figarski, Dariusz Paczewski, Lucjan Stapp, Sebastian Małyska.

Historia zmian

Wersja	Data	Uwagi
0.1	2011.10.08	Wersja beta dokumentu do pierwszej publikacji.
1.0	2012.01.11	Dokument zatwierdzony przez Zarząd SJSI. Bez zmian merytorycznych w stosunku do wersji 0.1.

Spis treści

Historia zmian 3
Spis treści 4
0. Wstęp 13
0.1 International Software Testing Qualifications Board 13
Cel dokumentu 13
Certyfikowany Tester – Poziom Zaawansowany 13
Poziom wiedzy 14
Egzamin 14
Akredytacja 14
Poziom szczegółowości 15
Struktura syllabusa 15
Terminy i definicje 15
Strategia 15
0.2 Oczekiwane rezultaty 16
0.2.1 Kierownik Testów – Poziom Zaawansowany 16
0.2.2 Analityk Testów – Poziom Zaawansowany 16
0.2.3 Techniczny Analityk Testów – Poziom Zaawansowany 17
0.3 Cele nauczania / poziom wiedzy 17
0.4 Cele nauczania dla modułu Kierownik Testów 20
0.5 Cele nauczania dla modułu Analityk Testów 27
0.6 Cele nauczania dla modułu Techniczny Analityk Testów 30
1. Podstawowe aspekty testowania oprogramowania 36
Terminologia 36
1.1 Wprowadzenie 36
1.2 Testowanie w cyklu życia oprogramowania 36
1.3 Systemy specyficzne 38
1.3.1 Systemy systemów (ang. *system of systems*) 38
1.3.1.1 Zarządzanie i testowanie systemu systemów 39

1.3.1.2 Charakterystyka cyklu życia systemu systemów ... 39
1.3.2 Systemy krytyczne ze względu na bezpieczeństwo ... 39
1.3.2.1 Zgodność z regulacjami ... 40
1.3.2.2 Systemy krytyczne ze względu na bezpieczeństwo i złożoność ... 40
1.4 Metryki i miary ... 41
1.5 Etyka ... 41
2. Procesy testowe ... 43
Terminologia: ... 43
2.1 Wprowadzenie ... 43
2.2 Modele procesu testowania ... 43
2.3 Planowanie i kontrola testów ... 44
2.4 Analiza i projektowanie testów ... 45
2.4.1 Identyfikacja warunków testowych ... 45
2.4.2 Tworzenie przypadków testowych ... 45
2.5 Implementacja i wykonanie testów ... 47
2.5.1 Implementacja testów ... 47
2.5.2 Wykonanie testów ... 48
2.6 Ocena spełnienia kryteriów wyjścia oraz raportowanie ... 50
2.7 Czynności na zakończenie testowania ... 51
3. Zarządzanie testowaniem ... 53
Terminologia ... 53
3.1 Wprowadzenie ... 53
3.2 Dokumentacja służąca do zarządzania testowaniem ... 53
3.2.1 Polityka testów ... 54
3.2.2 Strategia testów ... 54
3.2.3 Główny plan testów ... 56
3.2.4 Jednopoziomowy plan testów ... 57
3.3 Szablony dokumentacji planu testów ... 57
3.4 Szacowanie testów ... 57
3.5 Harmonogramowanie planowania testów ... 59
3.6 Monitorowanie i kontrola postępu testów ... 60

3.7 Biznesowa wartość testowania ... 62
3.8 Testowanie rozproszone, wykonywane przez osoby zakontraktowane oraz przez zewnętrzną firmę ... 63
3.9 Testowanie na podstawie ryzyka ... 63
3.9.1 Wstęp do testowania na podstawie ryzyka ... 63
3.9.2 Zarządzanie ryzykiem ... 65
3.9.2.1 Identyfikacja ryzyka ... 65
3.9.2.2 Analiza ryzyka ... 66
3.9.2.3 Łagodzenie ryzyka ... 68
Strategie zapobiegania ... 68
Łagodzenie ryzyka projektowego ... 68
Zapobieganie ryzyku produktowemu ... 69
Dopasowanie testowania do kolejnych cykli testowania ... 69
3.9.3 Zarządzanie ryzykiem w cyklu życia ... 70
3.10 Analiza przyczyn i skutków awarii ... 71
3.10.1 Obszary zastosowania ... 71
3.10.2 Fazy implementacji ... 71
3.10.3 Korzyści i koszty ... 71
3.11 Zastosowanie zarządzania testowaniem w różnych obszarach ... 72
3.11.1 Zarządzanie testami a testowanie eksploracyjne ... 72
3.11.2 Zarządzanie testowaniem a systemy systemów ... 73
3.11.3 Zarządzanie testowaniem a systemy krytyczne pod względem bezpieczeństwa ... 73
3.11.4 Inne zagadnienia dotyczące zarządzania testowaniem ... 74
3.11.4.1 Wymagania interesariuszy ... 74
3.11.4.2 Niezbędne wsparcie narzędziowe ... 75
3.11.4.3 Wymagana platforma sprzętowa ... 75
3.11.4.4 Zagadnienia organizacyjne ... 76
3.11.4.5 Zagadnienia dotyczące komunikacji ... 76
3.11.4.6 Zagadnienia bezpieczeństwa danych ... 76
4. Techniki Testowania ... 77
Terminologia ... 77
4.1 Wprowadzenie ... 77

4.2 Techniki oparte na specyfikacji ... 78
4.3 Techniki oparte o strukturę ... 80
Analiza pokrycia ... 82
4.4 Techniki oparte na defektach i techniki oparte na doświadczeniu ... 83
4.4.1 Techniki oparte na defektach ... 83
4.4.2 Techniki oparte na doświadczeniu ... 83
4.5 Analiza statyczna ... 86
4.5.1 Analiza statyczna kodu ... 86
4.5.1.1 Analiza przepływu sterowania ... 87
4.5.1.2 Analiza przepływu danych ... 87
4.5.1.3 Testowanie zgodności ze standardami programowania ... 87
4.5.1.4 Generowanie metryk kodu ... 87
4.5.2 Analiza statyczna architektury ... 87
4.5.2.1 Analiza statyczna strony webowej ... 87
4.5.2.2 Grafy wywołań ... 88
4.6 Analiza dynamiczna ... 88
4.6.1 Wprowadzenie ... 88
4.6.2 Wykrywanie wycieków pamięci ... 89
4.6.3 Wykrywanie dzikich wskaźników ... 89
4.6.4 Analiza wydajności ... 90
5. Testowanie właściwości oprogramowania ... 91
Terminologia ... 91
5.1 Wstęp ... 91
5.2 Atrybuty jakościowe do testowania dziedzinowego ... 91
5.2.1 Testowanie dokładności (ang. *accuracy testing*) ... 92
5.2.2 Testowanie odpowiedniości (ang. *suitability testing*) ... 92
5.2.3 Testowanie współdziałania (ang. *interoperability testing*) ... 92
5.2.4 Funkcjonalne testowanie zabezpieczeń (ang. *functional security testing*) ... 93
5.2.5 Testowanie użyteczności (ang. *usability testing*) ... 93
5.2.5.1 Specyfikacja testów użyteczności ... 94
Inspekcja, ocena lub przegląd ... 94

Walidacja wdrożenia 94
Ankiety i kwestionariusze 95
5.2.6 Testowanie dostępności (ang. *accessibility testing*) 95
5.3 Atrybuty jakościowe do testowania technicznego 95
5.3.1 Techniczne testowanie zabezpieczeń (ang. *technical security testing*) 96
5.3.1.1 Specyfikacja testów zabezpieczeń 97
5.3.2 Testowanie niezawodności (ang. *reliability testing*) 98
5.3.2.1 Testowanie odporności (ang. *tests for robustness*) 99
5.3.2.2 Testy odtwarzalności (ang. *recoverability testing*) 99
5.3.2.3 Specyfikacja testów niezawodności 100
5.3.3 Testowanie efektywności (ang. *efficiency testing*) 100
5.3.3.1 Testowanie wydajnościowe (ang. *performance testing*) 100
5.3.3.2 Testowanie obciążeniowe (ang. *load testing*) 101
5.3.3.3 Testowanie przeciążające (ang. *stress testing*) 101
5.3.3.4 Testy skalowalności (ang. *scalability testing*) 101
5.3.3.5 Test wykorzystania zasobów (ang. *test of resource utilization*) 102
5.3.3.6 Specyfikacja testów efektywności 102
5.3.4 Testowanie pielęgnowalności (ang. *maintainability testing*) 102
5.3.4.1 Dynamiczne testowanie pielęgnowalności 102
5.3.4.2 Analizowalność (pielęgnacja korekcyjna) 103
5.3.4.3 Zmienialność, stabilność i testowalność (pielęgnacja adaptacyjna) 103
5.3.5 Testowanie przenaszalności (ang. *portability testing*) 103
5.3.5.1 Testowanie instalowalności (ang. *installability testing*) 103
5.3.5.2 Testowanie koegzystencji (ang. *co-existence*) 104
5.3.5.3 Testowanie zdolności adaptacyjnej (ang. *adaptability testing*) 104
5.3.5.4 Testowanie zastępowalności (ang. *replaceability testing*) 105
6. Przeglądy 106
Terminologia 106
6.1 Wprowadzenie 106
6.2 Zasady przeprowadzania przeglądów 106
6.3 Rodzaje przeglądów 107

6.3.1 Przegląd kierowniczy i audyt ... 107
6.3.2 Przeglądy poszczególnych produktów projektu ... 108
6.3.3 Przeprowadzanie formalnych przeglądów ... 109
6.4 Wdrażanie przeglądów ... 109
6.5 Czynniki sukcesu w kontekście przeglądów ... 109
Czynniki techniczne: ... 109
Czynniki organizacyjne: ... 110
Czynniki ludzkie: ... 110
7. Zarządzanie incydentami ... 111
Terminologia ... 111
7.1 Wprowadzenie ... 111
7.2 Kiedy defekt może zostać znaleziony ... 111
7.3 Cykl życia defektu ... 111
7.3.1 Krok 1: Rozpoznanie ... 112
7.3.2 Krok 2: Badanie ... 112
7.3.3 Krok 3: Działanie ... 112
7.3.4 Krok 4: Dyspozycja ... 113
7.4 Atrybuty defektu ... 113
7.5 Metryki i zarządzanie incydentami ... 113
7.6 Komunikowanie incydentów ... 113
8. Standardy & Udoskonalanie Procesu Testowego ... 115
Terminologia ... 115
8.1 Wprowadzenie ... 115
8.2 Uznane standardy ... 115
8.2.1 Ogólne zagadnienia związane ze standardami ... 116
8.2.1.1 Źródła standardów ... 116
8.2.1.2 Przydatność standardów ... 116
8.2.1.3 Zgodność i konflikty ... 116
8.2.2 Standardy międzynarodowe ... 116
8.2.2.1 ISO ... 116
8.2.2.2 IEEE ... 117

8.2.3 Standardy narodowe 117
8.2.4 Standardy dziedzinowe 118
8.2.4.1 Lotnictwo 118
8.2.4.2 Przemysł kosmiczny 119
8.2.4.3 Żywność i leki 119
8.2.5 Inne standardy 119
8.3 Usprawnianie procesu testowania 120
8.3.1 Wstęp do doskonalenia procesów 120
8.3.2 Typy doskonalenia procesów 121
8.4 Doskonalenie procesu testowania 121
8.5 Doskonalenie procesu testowania z użyciem modelu TMM 123
8.6 Doskonalenie procesu testowania z użyciem modelu TPI 124
8.7 Doskonalenie procesu testowania z użyciem modelu CTP 125
8.8 Doskonalenie procesu testowania z użyciem modelu STEP 126
8.9 Capability Maturity Model Integration, CMMI 127
9. Narzędzia testowe i automatyzacja testów 128
Terminologia 128
9.1 Wprowadzenie 128
9.2 Koncepcje związane z narzędziami testowymi 128
9.2.1 Korzyści finansowe i ryzyko narzędzi testowych i automatyzacji 129
9.2.2 Strategie narzędzi testowych 130
9.2.3 Integracja i wymiana informacji pomiędzy narzędziami 130
9.2.4 Języki automatyzacji: skrypty, języki skryptowe 131
9.2.5 Wyrocznie testowe 131
9.2.6 Wdrażanie narzędzi testowych 132
9.2.7 Użycie narzędzi otwartych „Open Source” 133
9.2.8 Rozwój własnych narzędzi 133
9.2.9 Klasyfikacja narzędzi testowych 133
9.3 Kategorie narzędzi testowych 134
9.3.1 Narzędzia do zarządzania testami 134
9.3.2 Narzędzia do wykonywania testów 135

9.3.3 Narzędzia do debagowania i rozwiązywania problemów 136
9.3.4 Narzędzia do posiewu usterek i wstrzykiwania błędów 136
9.3.5 Symulatory i emulatory 137
9.3.6 Narzędzia do analizy statycznej i dynamicznej 137
9.3.6.1 Narzędzia do analizy statycznej 137
9.3.6.2 Narzędzia do analizy dynamicznej 138
9.3.7 Automatyzacja oparta o słowa kluczowe 138
9.3.8 Narzędzia do testów wydajnościowych 139
9.3.9 Narzędzia webowe 140
10. Umiejętności interpersonalne – skład zespołu 141
Terminologia 141
10.1 Wprowadzenie 141
10.2 Umiejętności indywidualne 141
10.3 Dynamika zespołu testowego 142
10.4 Dopasowanie testowania do organizacji 142
10.5 Motywacja 144
10.6 Komunikacja 144
11. Odnośniki 146
11.1 Standardy 146
11.1.1 Podział pod względem rozdziałów 146
11.1.2 Podział alfabetyczny 146
11.2 Książki 148
11.3 Inne odnośniki 148
12. Dodatek A – Podstawa syllabusa 150
Kryteria wejściowe dla tego certyfikatu 150
13. Dodatek B – Informacja dla Czytającego 152
13.1 Komisje Egzaminacyjne 152
13.2 Kandydaci i Dostawcy Szkoleń 152
14. Dodatek C – Informacja dla Dostawców Szkoleń 153
14.1 Podział na moduły 153
14.2 Czas trwania szkoleń 153

14.2.1 Czas ze względu na moduł 153
14.2.2 Części wspólne 153
14.2.3 Źródła 153
14.3 Ćwiczenia praktyczne 154
15. Dodatek D – Zalecenia 155
15.1 Zalecenia dla przemysłu 155

0. Wstęp

0.1 International Software Testing Qualifications Board

International Software Testing Qualifications Board (dalej zwany ISTQB®) jest organizacją składającą się z organizacji członkowskich, reprezentujących kraje lub regiony z całego świata. W momencie publikacji tej wersji syllabusa, ISTQB® liczyło 33 organizacje członkowskie[1]. Więcej szczegółów na temat struktury i członkostwa w ISTQB® można znaleźć na stronie www.istqb.org

Cel dokumentu

Niniejszy syllabus stanowi podstawę do międzynarodowej certyfikacji w zakresie testowania oprogramowania na poziomie zaawansowanym.

Odbiorcami tego dokumentu są:

1. Przedstawiciele organizacji narodowych[2], uprawnieni do przetłumaczenia syllabusa na język lokalny i akredytacji firm organizujących szkolenia. W tłumaczeniu syllabusa uwzględnione są wszelkie modyfikacje związane z dostosowaniem treści do wymogów języka docelowego. Organizacje narodowe mają prawo do zmiany listy referencji, dostosowując je do dostępnych na rynku publikacji.
2. Komisje egzaminacyjne (przynależące do organizacji narodowych)[3], odpowiedzialne za przygotowanie pytań egzaminacyjnych w języku lokalnym danego kraju.
3. Firmy szkoleniowe przygotowujące materiały szkoleniowe i dokonujące wyboru zalecanych metod nauczania.
4. Osoby przygotowujące się do egzaminu (w ramach akredytowanego kursu lub samodzielnie).
5. Międzynarodowa społeczność zajmująca się inżynierią oprogramowania, w celu zwiększenia prestiżu zawodu testera, a także w celu stworzenia bazy dla książek i artykułów dotyczących tych dziedzin.

Organizacje, które nie zostały wymienione powyżej, mogą używać niniejszego dokumentu do innych celów po uzyskaniu pisemnej zgody od ISTQB®.

Certyfikowany Tester – Poziom Zaawansowany

Certyfikat Certyfikowany Tester – Poziom Zaawansowany skierowany jest zarówno do osób mających duże doświadczenie w pracy na stanowisku związanym z testowaniem oprogramowania (testerzy, analitycy testowi, inżynierowie testów, konsultanci, kierownicy testów, testerzy akceptacyjni oraz programiści), jak również do osób pragnących pogłębić wiedzę z zakresu testowania oprogramowania (np. kierownicy projektów, menedżerowie jakości, kierownicy zespołów wytwarzania oprogramowania, analitycy biznesowi, dyrektorzy działów IT i konsultanci). Warunkiem otrzymania

[1] Aktualne informacje o składzie ISTQB można znaleźć na stronie http://istqb.org/display/ISTQB/ISTQB+Worldwide?atl_token=xTYMOtyqB0 [przypis SJSI]
[2] W Polsce organizacją narodową reprezentującą ISTQB® jest Stowarzyszenie Jakości Systemów Informatycznych (w skrócie SJSI, www.sjsi.org) [przypis SJSI]
[3] W Polsce jest to Komisja Egzaminacyjna SJSI [przypis SJSI].

certyfikatu poziomu zaawansowanego jest posiadanie certyfikatu Certyfikowany Tester – Poziom Podstawowy (ang. *ISTQB Certified Tester – Foundation Level*). Ponadto kandydaci powinni wykazać przed komisją egzaminacyjną, że posiadają wystarczające doświadczenie praktyczne wymagane na poziomie zaawansowanym. Szczegółowe informacje nt. wymaganego doświadczenia można uzyskać od komisji egzaminacyjnej.

Poziom wiedzy

Cele nauczania w poszczególnych rozdziałach zorganizowane są w taki sposób, aby mogły samodzielnie stanowić podstawę do certyfikacji dla poszczególnych modułów. Dalsze szczegóły i przykłady celów nauczania podane są w sekcji 0.3.

Zawartość syllabusa, używane pojęcia i najważniejsze elementy (cele) każdego podanego standardu, muszą być co najmniej zapamiętane (K1), nawet jeśli nie jest to jednoznacznie określone w celach nauczania.

Egzamin

Podstawą wszystkich egzaminów poziomu zaawansowanego jest niniejszy dokument oraz syllabus poziomu podstawowego. Odpowiedzi na pytania egzaminacyjne mogą wymagać rozumienia i umiejętności powiązania materiału z obu syllabusów.

Forma przeprowadzenia egzaminu opisana jest przez ISTQB® w wytycznych do egzaminu poziomu zaawansowanego, które w razie konieczności mogą być dostosowywane przez organizacje narodowe ISTQB®.

Egzamin może być częścią akredytowanego kursu (tzw. egzamin zamknięty) lub może być przeprowadzany niezależnie (bez kursu, tzw. egzamin otwarty). Egzaminy mogą przyjąć formę papierową lub elektroniczną, jednakże wszystkie muszą być przeprowadzone/nadzorowane przez osobę wyznaczoną przez organizację narodową ISTQB® lub komisję egzaminacyjną.

Akredytacja

Każda organizacja narodowa ISTQB® może akredytować firmy szkoleniowe posługujące się materiałami szkoleniowymi zgodnymi z tym syllabusem. Informacje dotyczące procesu akredytacji udostępniane są przez organizację narodową. Za kurs akredytowany uważa się taki kurs, który jest zgodny z niniejszym syllabusem i w ramach którego może odbyć się egzamin na certyfikat ISTQB®.

Dalsze wskazówki dla firm szkoleniowych znajdują się w Dodatku C – Wytyczne dla Dostawców Szkoleń.

Poziom szczegółowości

Poziom szczegółowości syllabusa umożliwia spójne nauczanie i egzaminowanie w skali międzynarodowej. W tym celu syllabus zawiera:

- ogólne cele nauczania opisujące założenia certyfikacji dla poziomu zaawansowanego
- cele nauczania dla każdego obszaru wiedzy, zawierające opis oczekiwanych wyników nauczania
- niezbędne informacje, w tym opis oraz odniesienia do dodatkowej literatury (jeśli jest to wymagane)
- listę pojęć, które muszą zostać zrozumiane i zapamiętane
- opisy kluczowych koncepcji, wraz z przyjętą literaturą i standardami

Zawartość syllabusa nie stanowi opisu całej wiedzy z zakresu testowania oprogramowania, a jedynie odnosi się do szczegółowości wymaganej na poziomie zaawansowanym.

Struktura syllabusa

Syllabus składa się z dziesięciu głównych rozdziałów. Każdy rozdział zawiera wprowadzenie, w którym opisano jego odniesienie do poszczególnych modułów[4].

Dla celów szkoleniowych, w podrozdziałach 0.3-0.6 podano szczegółowe cele nauczania dla każdego modułu. Wskazany jest tam również minimalny czas, jaki należy przeznaczyć na przećwiczenie opisanych tam zagadnień.

Zaleca się, aby czytając wybrany rozdział syllabusa jednocześnie analizować jego cele nauczania. Umożliwi to czytającemu pełne zrozumienie, co w ramach danego rozdziału jest konieczne i kluczowe dla każdego modułu.

Terminy i definicje

Wiele terminów w literaturze dotyczącej testowania oprogramowania używanych jest w sposób zamienny. Definicje wykorzystane w niniejszym syllabusie opisane zostały w „Słowniku wyrażeń związanych z testowaniem” (ang. *Standard Glossary of Terms Used in Software Testing*). Słownik został opublikowany przez ISTQB®, a przetłumaczony przez Stowarzyszenie Jakości Systemów Informatycznych (SJSI)[5].

Strategia

Istnieje wiele rodzajów strategii testowania np. strategie bazujące na specyfikacji, strukturze kodu, danych, analizie ryzyka, procesach, standardach czy taksonomiach. Wsparcia dla procesów testowania dostarczają różne procesy i narzędzia; dostępne są również metody ulepszania istniejących procesów.

[4] Wyodrębniono 3 moduły: Kierownik Testów, Analityk Testów, Techniczny Analityk Testów [przypis SJSI].
[5] Na dzień publikacji tego syllabusa obowiązującą wersją słownika był słownik w wersji 2.01 z 2010 roku. Więcej informacji na stronie www.sjsi.org [przypis SJSI].

Syllabus poziomu zaawansowanego jest stworzony w oparciu o podejścia proponowane w standardzie ISO 9126, z wyodrębnieniem podejść: funkcjonalnych, niefunkcjonalnych i pomocniczych. Jednocześnie wspomniane są również procesy wspomagające i metody usprawnień. Wybór takiej organizacji i procesów został przeprowadzony arbitralnie, by zapewnić słuszną, opartą na faktach bazę wiedzy dla testerów i kierowników testów poziomu zaawansowanego.

0.2 Oczekiwane rezultaty

Opisana w niniejszym syllabusie certyfikacja poziomu zaawansowanego będzie analizowana z uwzględnieniem trzech ról, z których każda reprezentuje podstawowe obowiązki i oczekiwania w organizacji. W każdej organizacji obowiązki i związane z nimi zadania mogą być rozdzielone pomiędzy różne osoby, bądź powierzone tylko jednej osobie. Role, o których tutaj mowa, przedstawiono poniżej[6].

0.2.1 Kierownik Testów – Poziom Zaawansowany

Osoby odpowiedzialne za zarządzanie testowaniem powinny umieć:

- definiować ogólne cele testowania i strategie testów dla testowanego systemu
- planować i śledzić wykonanie poszczególnych zadań
- opisywać i organizować odpowiednie czynności z zakresu testowania
- wybierać, pozyskiwać i przydzielać odpowiednie zasoby do zadań
- tworzyć, organizować i zarządzać zespołem testowym
- organizować kanały komunikacji pomiędzy członkami zespołów testowych, oraz pomiędzy zespołami testowymi a innymi zainteresowanymi stronami (np. pozostałymi członkami zespołu projektowego, przedstawicielami klienta itd)
- uzasadniać podjęte decyzje i raportować odpowiednie informacje we właściwym czasie

0.2.2 Analityk Testów – Poziom Zaawansowany

Analityk Testów powinien umieć:

- organizować zadania określone w strategii testowania pod względem wymagań biznesowych
- analizować system dostatecznie szczegółowo, aby sprostać oczekiwaniom użytkownika co do jakości
- oceniać wymagania systemowe pod kątem ich zasadności biznesowej
- planować i realizować odpowiednie czynności projektowe oraz raportować ich status
- dostarczać niezbędne dowody i metryki wspierające ocenę
- wdrażać niezbędne narzędzia i techniki, aby osiągnąć zdefiniowane cele

[6] W oryginale przy nazwie każdej z ról pojawia się uszczegółowienie, że chodzi o Poziom Zaawansowany. Podczas tłumaczenia przyjęto, że cały syllabus dotyczy certyfikacji poziomu zaawansowanego i w dalszej części dokumentu pominięto to uzupełnienie [przypis SJSI].

0.2.3 Techniczny Analityk Testów – Poziom Zaawansowany

Techniczny Analityk Testów powinien potrafić:

- organizować zadania zdefiniowane w strategii testów pod względem wymagań technicznych
- analizować wewnętrzną strukturę systemu dostatecznie szczegółowo, aby osiągnąć oczekiwany poziom jakości
- oceniać system pod kątem technicznych atrybutów jakości, takich jak wydajność, bezpieczeństwo, itp.
- planować i realizować odpowiednie czynności projektowe oraz raportować ich status
- testować produkt pod kątem technicznym
- dostarczać niezbędne dowody i metryki wspierające ocenę
- wdrażać niezbędne narzędzia i techniki, aby osiągnąć zdefiniowane cele

0.3 Cele nauczania / poziom wiedzy

Poniżej zostały wymienione cele nauczania mające zastosowanie w tym syllabusie. Każdy obszar tematyczny syllabusa będzie rozpatrywany zgodnie z przypisanymi do niego celami nauczania.

Poziom 1: Zapamiętaj (K1)

Kandydat rozpozna, zapamięta i powtórzy pojęcie lub koncepcję.

Słowa kluczowe: Zapamiętać, powtórzyć, rozpoznać, wiedzieć.

Przykład:

Kandydat potrafi rozpoznać definicję „awarii” także jako:

- „niedostarczenie usługi użytkownikowi końcowemu lub innemu odbiorcy”
  lub
- „niezgodność faktycznego działania modułu lub systemu z jego oczekiwanym zachowaniem lub rezultatem”.

Poziom 2: Zrozum (K2)

Kandydat potrafi podać powody lub wytłumaczyć dane zagadnienie; potrafi również podsumować, rozróżnić, sklasyfikować i podać przykłady dla omawianych zagadnień (np. porównanie terminów), koncepcji testowych, czy procedur testowych (wyjaśniając kolejność zadań).

Słowa kluczowe: Podsumowywać, klasyfikować, porównywać, przypisywać, przeciwstawiać, podawać przykłady, interpretować, tłumaczyć, reprezentować, dedukować, wnioskować, kategoryzować.

Przykłady:

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

- by znaleźć defekty, w czasie, gdy ich usunięcie jest tańsze
- by znaleźć najważniejsze defekty w pierwszej kolejności

Objaśnij podobieństwa i różnice między testowaniem integracyjnym a testowaniem 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 koncentrują się na aspektach działania systemu jako całości, pełnych procesach biznesowych („end-to-end").

Poziom 3: Zastosuj (K3)

Kandydat wie, jak poprawnie zastosować daną koncepcję lub technikę w odpowiednim kontekście. K3 na ogół stosuje się do wiedzy proceduralnej. Nie ma tutaj twórczego działania takiego, jak ocena aplikacji, lub tworzenie modelu dla istniejącego oprogramowania. Z poziomem K3 mamy do czynienia, kiedy posiadając określony model, stosujemy zawarte w syllabusie procedury do stworzenia przypadków testowych w oparciu o ten model.

Słowa kluczowe: Zastosować, wykonać, użyć, postępować zgodnie z procedurą, zastosować procedurę.

Przykład:

Kandydat potrafi:

- zdefiniować wartości graniczne dla poprawnych/ważnych i niepoprawnych/nieważnych danych podzielonych na klasy równoważności.
- użyć ogólnej procedury w celu określenia przypadków testowych (i zbiorów przypadków testowych) na podstawie diagramu stanów, w taki sposób, by pokryć wszystkie przejścia pomiędzy stanami.

Poziom 4: Przeanalizuj (K4)

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.

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

Przykład:

Kandydat potrafi:

- analizować ryzyko produktowe i zaproponować działania prewencyjne i naprawcze
- opisywać, które elementy zgłoszenia incydentu są obiektywne a które wynikają z otrzymanych wyników.

Literatura dotycząca poziomów poznawczych celów nauczania:

Bloom, B. S. (1956). *Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain,* David McKay, Co. Inc.

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

0.4 Cele nauczania dla modułu Kierownik Testów

Ta sekcja dostarcza listę celów nauczania dla modułu Kierownik Testów.

Zasadniczo wszystkie części tego syllabusa mogą być egzaminowane na poziomie K1. Tym samym kandydat powinien umieć rozpoznać, zapamiętać i powtórzyć pojęcie lub teorię. Poniższe zestawienie zawiera tylko cele nauczania na poziomie K2, K3 i K4.

Wstęp – (60 minut)

(włączając powtórkę z syllabusa poziomu podstawowego ISTQB®)

Rozdział 1: Podstawowe aspekty testowania oprogramowania – (150 minut)

1.2 Testowanie w cyklu życia oprogramowania

Kandydat potrafi:

- (K2) opisać, w jaki sposób testowanie związane jest z czynnościami wytwarzania i utrzymania oprogramowania.
- (K4) przeanalizować modele cyklu życia oprogramowania i wskazywać najbardziej odpowiednie dla nich zadania/czynności testowe (należy tu rozróżnić pomiędzy czynnościami testowymi i programistycznymi).

1.3 Systemy specyficzne

Kandydat potrafi:

- (K2) wyjaśnić na przykładach specyfikę testowania systemu systemów (ang. *system of* systems).
- (K2) wyjaśnić najważniejsze aspekty testowania systemów krytycznych ze względu na bezpieczeństwo, celem wykazania zgodności z regulacjami.

1.4 Metryki i miary

Kandydat potrafi:

- (K2) opisać i porównać standardowe metryki testowe.
- (K3) monitorować przebieg testowania poprzez pomiar przedmiotu testowego oraz procesu testowego.

Rozdział 2: Procesy testowe – (120 minut)

2.3 Planowanie i kontrola testów

Kandydat potrafi:

- (K2) opisać na przykładach, jak strategie testowe wpływają na planowanie testów.
- (K2) porównać produkty powstające w wyniku testowania i wyjaśnić na przykładach powiązania tych produktów z produktami powstającymi w ramach prac programistycznych.
- (K2) sklasyfikować czynności monitorujące, służące do określenia, czy misja, strategie i cele testów zostały osiągnięte/zrealizowane.

2.5 Implementacja i wykonanie testów

Kandydat potrafi:

- (K2) wyjaśnić warunki wstępne konieczne do wykonania testów.
- (K2) wyjaśnić, na przykładach, zalety i wady wczesnego testowania z uwzględnieniem różnych technik testowania.
- (K2) wyjaśnić powody, dla których użytkownik i/lub klient może być zaangażowany w wykonanie testów.
- (K2) wyjaśnić, jak w zależności od poziomu testowania może zmieniać się poziom logowania testów.

2.6 Ocena spełnienia kryteriów wyjścia oraz raportowanie

Kandydat potrafi:

- (K2) podsumować dane zebrane podczas testowania, w celu odpowiedniego zaraportowania i oceny kryteriów wyjściowych

2.7 Czynności na zakończenie testowania

Kandydat potrafi:

- (K2) podsumować cztery grupy czynności zamykających testy.
- (K3) uogólnić zdobytą wiedzę w celu określenia, co powinno zostać powtórzone i udoskonalone.

Rozdział 3: Zarządzanie testowaniem – (1120 minut)

3.2 Dokumentacja służąca do zarządzania testowaniem

Kandydat potrafi:

- (K4) nakreślić dokumenty służące do zarządzania testowaniem takie jak plan testów, specyfikacja projektu testu, procedura testowa, zgodnie z IEEE 829.
- (K2) opisać co najmniej cztery ważne elementy dokumentacji strategii/podejścia do testów oraz przedstawić, które dokumenty zgodnie z IEEE 829 zawierają elementy strategii testów.

- (K2) przedstawić powody i metody odejścia od strategii testów oraz sposoby ich udokumentowania w innej dokumentacji zarządzania testami.

3.3 Szablony dokumentacji planu testów

Kandydat potrafi:

- (K2) streścić struktury głównego planu testów wg IEEE 829
- (K2) dopasować zawartość planu testów proponowaną w IEEE 829 do specyfiki organizacji, ryzyka produktowego, jak również do ryzyka, wielkości i stopnia sformalizowania projektu.

3.4 Szacowanie testów

Kandydat potrafi:

- (K3) oszacować pracochłonności czynności testowych dla przykładowego, małego systemu, z wykorzystaniem metryk opartych o doświadczenie, uwzględniając czynniki mające wpływ na koszty i czas trwania testów.
- (K2) zrozumieć oraz zaprezentować przykłady czynników wymienionych w tym dokumencie, które mogą wpłynąć na niedokładność oszacowań pracochłonności prac testowych.

3.5 Harmonogramowanie planowania testów

Kandydat potrafi:

- (K2) objaśnić na przykładach korzyści płynące z wczesnego i iteracyjnego planowania testów.

3.6 Monitorowanie i kontrola postępu testów

Kandydat potrafi:

- (K2) porównać różne procedury nadzoru postępu prac w procesie testowym.
- (K2) podać przynajmniej 5 różnych przykładów wyjaśniających, w jaki sposób wnioski z monitorowania postępu prac w testach wpływają na przebieg procesu testowego.
- (K4) wykorzystać dane z monitorowania i kontroli postępu prac testowych oraz innych miar celem usprawnienia procesu testowego oraz przedstawić propozycje usprawnień.
- (K4) przeanalizować wyniki testów i określić postępy testów, udokumentowane w postaci raportu monitoringu oraz raportu końcowego testów, obejmującego wszystkie 4 wymiary raportowania.

3.7 Biznesowa wartość testowania

Kandydat potrafi:

- (K2) podać przykłady (miary) dla każdej z 4 kategorii określającej „koszt jakości”.
- (K3) wyszczególnić dla danego kontekstu ilościowe i/lub jakościowe wartości, które wnosi testowanie.

3.8 Testowanie rozproszone, wykonywane przez osoby zakontraktowane oraz przez zewnętrzną firmę

Kandydat potrafi:

- (K2) określić ryzyko, przedstawić podobieństwa i różnice pomiędzy 3 wymienionymi wyżej rodzajami zatrudnienia.

3.9 Testowanie na podstawie ryzyka

3.9.1 Wstęp do testowania na podstawie ryzyka

Kandydat potrafi:

- (K2) wyjaśnić różnorodne sposoby, jak testowanie w oparciu o ryzyko adresuje ryzyko.
- (K4) zidentyfikować ryzyka produktowe i projektowe oraz określić odpowiednią strategię testów i plan testów w oparciu o te ryzyka.

3.9.2 Zarządzanie ryzykiem

Kandydat potrafi:

- (K3) wykonać analizę ryzyka dla produktu z punktu widzenia testera, stosując podejście analizy przyczyn i skutków awarii (FMEA).
- (K4) zebrać i podsumować dane o ryzyku z różnych perspektyw (od różnych interesariuszy) oraz użyć tej wiedzy do określenia czynności testowych mających na celu ograniczenie ryzyka.

3.9.3 Zarządzanie ryzykiem w cyklu życia

Kandydat potrafi:

- (K2) opisać charakterystyczne cechy procesu zarządzania ryzykiem, które powodują, że powinien to być proces iteracyjny.
- (K3) przekształcić strategię testów utworzoną w oparciu o analizę ryzyka w konkretne czynności testowe i monitorować wyniki realizacji strategii podczas wykonywania testów.
- (K4) przeanalizować i stworzyć raport wyników testów oraz określić/zaproponować pozostałe ryzyka, tym samym umożliwiając kierownikom projektu podjęcie odpowiednich decyzji dotyczących wydania.

3.10 Analiza przyczyn i skutków awarii

Kandydat potrafi:

- (K2) opisać koncepcję FMEA oraz wyjaśnić na przykładach zastosowanie tej koncepcji w projektach i płynące z tego tytułu korzyści.

3.11 Zastosowanie zarządzania testowaniem w różnych obszarach

Kandydat potrafi:

- (K2) porównać związane ze strategią kwestie zarządzania testami w testowaniu eksploracyjnym, testowaniu systemu systemów i testowaniu systemów krytycznych ze względu na bezpieczeństwo, wskazać korzyści i wady, adekwatność i wpływ tych kwestii na planowanie, pokrycie oraz monitorowanie i kontrolę.

Rozdział 4: Techniki testowania – (0 minut)

Brak celów nauczania (na dowolnym K-poziomie) mających zastosowanie dla modułu Kierownik Testów.

Rozdział 5: Testowanie właściwości oprogramowania – (0 minut)

Brak celów nauczania (na dowolnym K-poziomie) mających zastosowanie dla modułu Kierownik Testów.

Rozdział 6: Przeglądy – (120 minut)

6.2 Zasady przeprowadzania przeglądów

Kandydat potrafi:

- (K2) wyjaśnić korzyści płynące z wykonywania przeglądów (w porównaniu z testami dynamicznymi i innymi statycznymi technikami testów).

6.4. Wdrażanie przeglądów

Kandydat potrafi:

- (K2) porównać typy przeglądów, wskazać ich mocne oraz słabe strony i obszar zastosowania.
- (K3) kierować zespołem wykonującym przegląd w formalnym przeglądzie zgodnie z określonymi krokami.
- (K4) nakreślić plan przeglądu jako część planu jakości/testów dla projektu uwzględniając techniki przeglądu, potencjalne defekty, umiejętności personelu i dostosować go do odpowiednich dynamicznych podejść testowych.

6.5 Czynniki sukcesu w kontekście przeglądów

Kandydat potrafi:

- (K2) przedstawić ryzyka dotyczące przeglądów, niezwiązane z czynnikami technicznymi, organizacyjnymi i ludzkimi.

Rozdział 7: Zarządzanie incydentami – (80 minut)

Kandydat potrafi:

- (K3) przeprowadzić proces zarządzania defektem zgodnie z procedurą zarządzania cyklem życia incydentu zaproponowaną w standardzie IEEE 1044-1993.
- (K3) przeprowadzić ocenę zgłoszeń defektów (w tym zasad ich opisu/atrybutów) w oparciu o standard IEEE 1044 -1993 celem poprawy jakości tych zgłoszeń.
- (K4) przeprowadzić analizę zgłoszeń defektów powstałych na przestrzeni danego okresu czasu oraz dokonać modyfikacji zasad opisu tych zgłoszeń.

Rozdział 8: Standardy i udoskonalanie procesu testowego – (120 minut)

Kandydat potrafi:

- (K2) podsumować źródła standardów dotyczących oprogramowania oraz wyjaśnić ich przydatność dla testowania oprogramowania

8.4 Doskonalenie procesu testowania

Kandydat potrafi:

- (K2) opracować plan doskonalenia testów przy zastosowaniu standardowych kroków i z uwzględnieniem właściwych osób.
- (K2) podsumować proces poprawy testowania zdefiniowany w TMM, TPI, CTP, STEP oraz w obszarach procesów weryfikacji i walidacji w CMMI.
- (K2) wyjaśnić kryteria oceny w modelach poprawy procesu testowego TMM, TPI, CTP, STEP oraz procesach weryfikacji i walidacji w CMMI.

Rozdział 9: Narzędzia testowe i automatyzacja testów – (90 minut)

9.2 Koncepcje związane z narzędziami testowymi

Kandydat potrafi:

- (K2) porównać elementy i aspekty każdej z koncepcji narzędzi testowych: "Korzyści finansowe i ryzyko narzędzi testowych i automatyzacji", "Strategie narzędzi testowych", "Integracja i wymiana informacji pomiędzy narzędziami", "Języki automatyzacji", "Wyrocznie

testowe", "Wdrażanie narzędzi testowych", "Użycie narzędzi *open-source*", "Rozwój własnych narzędzi" i "Klasyfikacja narzędzi testowych".
- (K2) wyjaśnić dlaczego i kiedy konieczne jest stworzenie strategii dotyczącej narzędzi testowych lub planu wdrożenia narzędzia testowego.
- (K2) wyjaśnić różne fazy wdrożenia narzędzia testowego.

9.3 Kategorie narzędzi testowych

Kandydat potrafi:

- (K2) przedstawić kategorie narzędzi testowych (cele, zastosowanie, mocne strony, ryzyka i przykłady).
- (K2) przedstawić i podsumować wymagania specyficzne dla narzędzi testowych i rozwiązań *open source* stosowanych w testowaniu systemów krytycznych ze względu na bezpieczeństwo.
- (K2) przedstawić istotne aspekty i konsekwencje wdrożenia różnych narzędzi testowych, ich użycie i wpływ na proces testowy.
- (K2) opisać, kiedy i dlaczego wdrażanie własnego narzędzia jest konieczne/uzasadnione oraz jakie są zyski, ryzyka i konsekwencje tej decyzji.

Rozdział 10 Umiejętności interpresonalne – skład zespołu (240 minut)

10.2 Umiejętności indywidualne

Kandydat potrafi:

- (K3) wykorzystać kwestionariusz oceny w celu określenia silnych i słabych stron członków zespołu (pod względem umiejętności korzystania z oprogramowania, posiadanej wiedzy dziedzinowej i biznesowej, wiedzy o różnych aspektach/obszarach wytwarzania systemów informatycznych, testowania oraz posiadanych umiejętności interpersonalnych).

10.3 Dynamika zespołu testowego

Kandydat potrafi:

- (K3) wykonać analizę potrzeb w celu uzupełnienia brakujących umiejętności technicznych i miękkich dla wakatów w organizacji.

10.4 Dopasowanie testowania do organizacji

Kandydat potrafi:

- (K2) opisać różne formy organizacyjne i porównać je z rozproszonym, zakontraktowanym (ang. *insourcing*), zewnętrznym (ang. *outsourcing*) wykonywaniem testów.

10.5. Motywacja

Kandydat potrafi:

- (K2) przedstawić przykłady czynników motywujących i demotywujących testerów.

10.6 Komunikacja

Kandydat potrafi:

- (K2) opisać na przykładach profesjonalną, obiektywną i efektywną komunikację w projekcie z perspektywy testera, z ewentualnym uwzględnieniem ryzyk i potencjalnych usprawnień.

0.5 Cele nauczania dla modułu Analityk Testów

Ta sekcja zawiera listę celów nauczania dla modułu Analityk Testów.

Zasadniczo wszystkie części tego syllabusa mogą być egzaminowane na poziomie K1. Tym samym kandydat powinien umieć rozpoznać, zapamiętać i powtórzyć pojęcie lub teorię. Poniższe zestawienie zawiera tylko cele nauczania na poziomie K2, K3 i K4.

Wstęp – (60 minut)

(włączając powtórkę z syllabusa poziomu podstawowego ISTQB®)

Rozdział 1: Podstawowe aspekty testowania oprogramowania – (30 minut)

Rozdział 2: Procesy testowe – (180 minut)

2.4 Analiza i projektowanie testów

Kandydat potrafi:

- (K2) wyjaśnić powody, dla których testy funkcjonalne przeprowadza się w określonych etapach w cyklu życia oprogramowania.
- (K2) wyjaśnić - na przykładach - kryteria, które wpływają na strukturę i poziom tworzenia warunków testowych.
- (K2) przedstawić, w jaki sposób analiza i projektowanie testów są techniką statyczną, którą można użyć do wykrywania defektów.
- (K2) wyjaśnić – na przykładach – koncepcję wyroczni testowej oraz sposób, w jaki wyrocznia testowa może być użyta w specyfikacji testów.

2.5 Implementacja i wykonanie testów

Kandydat potrafi:

- (K2) wyjaśnić warunki wstępne dla wykonania testów, włączając testalia, środowisko testowe, zarządzanie konfiguracją i zarządzanie defektami.

2.6 Ocena spełnienia kryteriów wyjścia oraz raportowanie

Kandydat potrafi:

- (K3) dla danego zbioru miar określić, czy zostały spełnione warunki zakończenia testów.

Rozdział 3: Zarządzanie testowaniem – (120 minut)

3.9.2 Zarządzanie ryzykiem

Kandydat potrafi:

- (K3) określić priorytety doboru przypadków testowych, analizy pokrycia testów oraz wyboru danych testowych w oparciu o ryzyko.
- (K2) udokumentować czynności planowania testów i procedur testowych w przypadku zastosowania podejścia opartego o zarządzanie ryzykiem.

Rozdział 4: Techniki testowania – (1080 minut)

4.2 Techniki oparte na specyfikacji

Kandydat potrafi:

- (K2) wymienić przykłady typowych defektów identyfikowanych za pomocą każdej techniki opartej na specyfikacji, zapewnić odpowiednie kryteria pokrycia testowego.
- (K3) napisać przypadki testowe dla danych modeli testowych używając następujących technik (test powinien osiągnąć zadany model pokrycia):
  - podział na klasy równoważności
  - analiza wartości brzegowych
  - tablica decyzyjna
  - testowanie przejść pomiędzy stanami
  - metoda drzewa klasyfikacji
  - testowanie par
  - przypadki użycia
- (K4) wykonać analizę systemu lub jego specyfikacji wymagań, w celu określenia, którą technikę opartą na specyfikacji stosować dla określonych celów; nakreślić specyfikację testów w oparciu o IEEE 829, ze szczególnym uwzględnieniem funkcjonalnych i dziedzinowych przypadków i procedur testowych.

4.4 Techniki oparte na defektach i techniki oparte na doświadczeniu

Kandydat potrafi:

- (K2) opisać podstawy i przyczyny używania technik opartych na defektach i wykazać różnice ich wykorzystania w odróżnieniu od technik opartych o specyfikację i strukturę.
- (K2) wyjaśnić na przykładach taksonomię defektów i jej użycie.
- (K2) wyjaśnić zasady i powody używania techniki opartej o doświadczenie.
- (K3) stworzyć specyfikację, wykonać i zaraportować wynik testów używając techniki eksploracyjnej.
- (K2) sklasyfikować defekty, jakie można zidentyfikować za pomocą różnych typów ataków na oprogramowanie, zastosowanych stosownie do defektów, które są celem tych ataków.
- (K4) przeprowadzić analizę systemu w celu określenia, jakie techniki (w oparciu o specyfikację, w oparciu o defekty, w oparciu o doświadczenie) są właściwe, by osiągnąć określone cele testowania.

Rozdział 5: Testowanie właściwości oprogramowania – (210 minut)

5.2 Atrybuty jakościowe do testowania dziedzinowego

Kandydat potrafi:

- (K4) wyjaśnić na przykładach, jakie techniki testowania opisane w rozdziale 4 są odpowiednie do testowania właściwości: poprawności, odpowiedniości, współdziałania, zabezpieczeń funkcjonalnych oraz charakterystyk dostępności.
- (K3) nakreślić, zaprojektować, wyspecyfikować oraz wykonać testy użyteczności w oparciu o odpowiednie techniki i pokrywając dane cele testowe i/lub defekty.

5.3 Atrybuty jakościowe do testowania technicznego

Kandydat potrafi:

- (K2) wyjaśnić przyczyny, dla których testy efektywności, niezawodności i bezpieczeństwa technicznego są włączone do strategii testowej oraz podać przykłady defektów, które w ten sposób mogą być znalezione.
- (K2) scharakteryzować typy testów niefunkcjonalnych w testowaniu technicznym poprzez określenie typowych znajdowanych (atakowanych) defektów, ich typowego zastosowania w cyklu życia oprogramowania oraz technik projektowania testów.

Rozdział 6: Przeglądy – (180 minut)

Kandydat potrafi:

- (K2) użyć listy kontrolnej do weryfikacji kodu i architektury systemu z perspektywy testera.
- (K3) użyć listy kontrolnej do weryfikacji wymagań i przypadków użycia z perspektywy testera.
- (K2) porównać typy przeglądów i wskazać ich słabe i silne strony oraz zakres użycia.

Rozdział 7: Zarządzanie incydentami – (120 minut)

Kandydat potrafi:

- (K4) wykonać analizę, klasyfikację i opisać defekty funkcjonalne i niefunkcjonalne w zgłoszeniach defektów.

Rozdział 8: Standardy i udoskonalanie procesu testowego – (0 minut)

Nie ma żadnych celów nauczania (na żadnym K-poziomie) mających zastosowanie dla modułu Analityk Testów.

Rozdział 9: Narzędzia testowe i automatyzacja – (90 minut)

9.2 Koncepcje związane z narzędziami testowymi

Kandydat potrafi:

- (K2) porównać elementy i aspekty każdej z koncepcji narzędzi testowych: "Korzyści finansowe i ryzyko narzędzi testowych i automatyzacji", "Strategie narzędzi testowych", "Integracja i wymiana informacji pomiędzy narzędziami", "Języki automatyzacji", "Wyrocznie testowe", "Wdrażanie narzędzi testowych", "Użycie narzędzi *open-source*", "Rozwój własnych narzędzi" i "Klasyfikacja narzędzi testowych".

9.3 Kategorie narzędzi testowych

Kandydat potrafi:

- (K2) podsumować (z podaniem przykładów) kategorie narzędzi testowych według: celów użycia, przeznaczenia, silnych stron oraz ryzyk.
- (K2) zmapować narzędzia z różnych kategorii do różnych poziomów i typów testów.

Rozdział 10: Umiejętności interpresonalne – skład zespołu – (30 minut)

10.6 Komunikacja

Kandydat potrafi:

- (K2) opisać na przykładach profesjonalną, obiektywną i efektywną komunikację w projekcie z perspektywy testera, ewentualnie rozważając ryzyka i możliwości.

0.6 Cele nauczania dla modułu Techniczny Analityk Testów

Ta sekcja dostarcza listę celów nauczania dla modułu Techniczny Analityk Testów.

Zasadniczo wszystkie części tego syllabusa mogą być egzaminowane na poziomie K1. Tym samym kandydat powinien umieć rozpoznać, zapamiętać i powtórzyć pojęcie lub teorię. Poniższe zestawienie zawiera tylko cele nauczania na poziomie K2, K3 i K4.

Wstęp – (60 minut)

(włączając powtórkę z syllabusa poziomu podstawowego ISTQB®)

Rozdział 1: Podstawowe aspekty testowania oprogramowania – (30 minut)

Rozdział 2: Procesy testowe – (180 minut)

2.4 Analiza i projektowanie testów

Kandydat potrafi:

- (K2) wyjaśnić etapy w cyklu życia oprogramowania, w których mogą być przeprowadzane testy niefunkcjonalne i testy oparte o architekturę; wyjaśnić powody, dla których testy niefunkcjonalne mają miejsce tylko na specyficznych etapach cyklu życia oprogramowania.
- (K2) podać (objaśnić na przykładach) kryteria, które wpływają na strukturę i poziom tworzenia warunków testowych.
- (K2) uzasadnić, że analiza i projektowanie testów są techniką statyczną, którą można użyć do wykrywania defektów.
- (K2) wyjaśnić – na przykładach – koncepcję wyroczni testowej oraz jak wyrocznia testowa może być użyta w specyfikacji testów.

2.5 Implementacja i wykonanie testów

Kandydat potrafi:

- (K2) wyjaśnić warunki wstępne dla wykonania testów, włączając testalia, środowisko testowe, zarządzanie konfiguracją i zarządzanie defektami.

2.6 Ocena spełnienia kryteriów wyjścia oraz raportowanie

Kandydat potrafi:

- (K3) dla danego zbioru miar określić, czy warunki zakończenia testów są spełnione.

Rozdział 3: Zarządzanie testowaniem – (120 minut)

3.9. Testowanie na podstawie ryzyka

Kandydat potrafi:

- (K2) określić czynności w testowaniu opartym o ryzyko dla planowania i wykonywania testów technicznych

Rozdział 4: Techniki testowania – (330 minut)

4.2 Techniki oparte na specyfikacji

Kandydat potrafi:

- (K2) wyszczególnić przykłady typowych defektów dla każdej techniki opartej na specyfikacji.
- (K2) na podstawie danego modelu oprogramowania napisać rzeczywiste przypadki testowe używając następujących technik projektowania testów (testy powinny osiągnąć zadany model pokrycia):
  - podział na klasy równoważności
  - analiza wartości brzegowych
  - tablica decyzyjna
  - testowanie przejść pomiędzy stanami
- (K4) wykonać analizę systemu lub jego specyfikacji wymagań, w celu określenia, którą technikę opartą na specyfikacji stosować dla określonych celów; określić specyfikację testów w oparciu o IEEE 829, ze szczególnym uwzględnieniem przypadków dla testów modułowych i niefunkcjonalnych oraz procedur testowych.

4.3 Techniki oparte o strukturę

Kandydat potrafi:

- (K2) wyszczególnić przykłady typowych defektów identyfikowanych przez każdą specyficzną technikę opartą o specyfikację.
- (K3) napisać rzeczywiste przypadki testowe używając następujących technik projektowania testów (testy powinny osiągnąć zadany model pokrycia):
  - testowanie instrukcji
  - testowanie decyzji
  - pokrycie warunków znaczących
  - testowanie wielokrotnych warunków
- (K4) wykonać analizę w celu określenia, które techniki oparte o strukturę nadają się do osiągnięcia specyficznych celów testowych.
- (K2) zrozumieć każdą technikę opartą o strukturę i odpowiadające jej kryteria pokrycia oraz to, kiedy warto jej używać.
- (K4) umieć porównać i przeanalizować, której techniki opartej o strukturę użyć w różnych sytuacjach.

4.4 Techniki oparte na defektach i techniki oparte na doświadczeniu

Kandydat potrafi:

- (K2) opisać podstawy i przyczyny używania techniki opartej o defekty i rozróżnić jej użycie od technik opartych o specyfikację i strukturę.
- (K2) wyjaśnić na przykładach i zastosować taksonomię defektów.
- (K2) zrozumieć zasady i powody używania techniki opartej o doświadczenie oraz tego, kiedy jej używamy.
- (K3) wyspecyfikować, wykonać i zaraportować testy używając techniki eksploracyjnej.
- (K2) wyspecyfikować testy używających różnych typów ataków na oprogramowanie, w zależności od defektów, które są celem tych ataków.
- (K4) wykonać analizę systemu w celu określenia, jakie techniki (w oparciu o specyfikację, w oparciu o defekty, w oparciu o doświadczenie) zostaną zastosowane, by osiągnąć odpowiednie cele testowania.

4.5 Analiza statyczna

Kandydat potrafi:

- (K3) użyć algorytmów „analiza przepływu sterowania", „analiza przepływu danych" do weryfikacji, czy kod nie zawiera anomalii związanych z przepływem sterowania lub z przepływem danych.
- (K4) ziterpretować wyniki procesu analizy przepływu sterowania i danych otrzymywanych z narzędzi w celu oszacowania, czy kod nie zawiera anomalii związanych z przepływem sterowania lub z przepływem danych.
- (K2) wyjaśnić zastosowanie grafu wywołań do oceny jakości architektury; wyjaśnienie powinno zawierać identyfikowane defekty, opis zastosowania w projektowaniu i planowaniu testów oraz ograniczenia.

4.6 Analiza dynamiczna

Kandydat potrafi:

- (K2) wyjaśnić sposób wykonywania analizy dynamicznej dla kodu i przedstawić defekty, które można zidentyfikować używając tej techniki, określając jej ograniczenia.

Rozdział 5: Testowanie właściwości oprogramowania – (240 minut)

5.2 Atrybuty jakościowe do testowania dziedzinowego

Kandydat potrafi:

- (K2) scharakteryzować niefunkcjonalne typy testów dla testowania dziedzinowego poprzez typowe defekty, typowe zastosowanie tych testów w cyklu życia oprogramowania i techniki testowe użyteczne do projektowania testów.
- (K4) wyspecyfikować przypadki testowe dla odpowiednich typów testów niefunkcjonalnych, pokrycia danego celu testowego i odkrywanych defektów.

5.3 Atrybuty jakościowe do testowania technicznego

Kandydat potrafi:

- (K2) scharakteryzować niefunkcjonalne typy testów dla testowania dziedzinowego poprzez typowe defekty, typowe zastosowanie tych testów w cyklu życia oprogramowania i techniki testowe użyteczne do projektowania testów.
- (K2) zrozumieć i wyjaśnić etapy w cyklu życia oprogramowania, gdzie mogą mieć zastosowanie testy bezpieczeństwa, niezawodności oraz efektywności (włączając odpowiadające pod-atrybuty z ISO 9126).
- (K2) rozróżnić pomiędzy typami defektów znajdowanymi podczas testów bezpieczeństwa, niezawodności oraz efektywności (włączając odpowiadające pod-atrybuty z ISO 9126).
- (K2) określić warunki testowania do testów atrybutów jakościowych: bezpieczeństwo, niezawodność oraz efektywność i odpowiadających pod-atrybutów z ISO 9126.
- (K2) zrozumieć i wyjaśnić powody, dla których testy pielęgnacji, przenaszalności oraz dostępności powinny być dołączane do strategii testowej.
- (K3) określić warunki testowe dla pielęgnacji oraz przenaszalności w testach niefunkcjonalnych.

Rozdział 6: Przeglądy – (180 minut)

Kandydat potrafi:

- (K4) nakreślić listę kontrolną dla przeglądu mającego na celu znalezienie typowych defektów podczas przeglądu kodu i architektury.
- (K3) porównać typy przeglądów i wskazać ich mocne i słabe strony oraz zakres wykorzystania.

Rozdział 7: Zarządzanie incydentami – (120 minut)

Kandydat potrafi:

- (K4) wykonać analizę, klasyfikację i opisać defekty funkcjonalne i niefunkcjonalne w zgłoszeniu defektu.

Rozdział 8: Standardy i udoskonalanie procesu testowego – (120 minut)

Nie ma żadnych celów (na żadnym K-poziomie) mających zastosowanie dla modułu Techniczny Analityk Testów.

Rozdział 9: Narzędzia testowe i automatyzacja – (210 minut)

9.2 Koncepcje związane z narzędziami testowymi

Kandydat potrafi:

- (K2) porównać elementy i aspekty każdej z koncepcji narzędzi testowych: "Korzyści finansowe i ryzyko narzędzi testowych i automatyzacji", "Strategie narzędzi testowych", "Integracja i wymiana informacji pomiędzy narzędziami", "Języki automatyzacji", "Wyrocznie testowe", "Wdrażanie narzędzi testowych", "Użycie narzędzi *open-source*", "Rozwój własnych narzędzi" i "Klasyfikacja narzędzi testowych".

9.3 Kategorie narzędzi testowych

Kandydat potrafi:

- (K2) podsumować (z podaniem przykładów) kategorie narzędzi testowych według: celów użycia, przeznaczenia, silnych stron oraz ryzyk.
- (K2) zmapować narzędzia z różnych kategorii do różnych poziomów i typów testów.

9.3.7 Automatyzacja oparta o słowa kluczowe

Kandydat potrafi:

- (K2) stworzyć tablice słów kluczowych/słów akcji przy zastosowaniu algorytmów do wyboru słów kluczowych, które mogą być użyte przez narzędzie wykonujące testy.
- (K3) wykonać adaptację skryptów testowych zarejestrowanych przez narzędzie rejestrująco – odtwarzające, by umożliwić automatyzację wykonania testów opartych na słowach kluczowych.

9.3.8 Narzędzia do testów wydajnościowych

Kandydat potrafi:

- (K3) zaprojektować testy wydajnościowe przy użyciu odpowiednich narzędzi, uwzględniając planowanie i pomiar charakterystyk systemu.

Rozdział 10: Umiejętności interpresonalne – skład zespołu – (30 minut)

10.6 Komunikacja

Kandydat potrafi:

- (K2) opisać na przykładach profesjonalną, obiektywną i efektywną komunikację w projekcie z perspektywy testera, ewentualnie rozważając ryzyka i możliwości.

1. Podstawowe aspekty testowania oprogramowania

Terminologia

Etyka, miary, metryka, systemy krytyczne ze względu na bezpieczeństwo, systemy systemów, cykl życia oprogramowania

1.1 Wprowadzenie

Rozdział jest wprowadzeniem do podstawowych zagadnień z obszaru testowania, mających fundamentalne znaczenie dla wszystkich osób profesjonalnie zajmujących się testami: kierowników testów, analityków testów, jak również dla technicznych analityków testów.

Objaśnienia tych tematów w kontekście wybranych modułów dokonają dostawcy szkoleń, jednocześnie podając odpowiednie przykłady. Np. w module Techniczny Analityk Testów, w ramach ogólnego zagadnienia „metryki i miary" (rozdział 1.4) użyte są przykłady specyficznych metryk technicznych, takich jak miary wydajności.

W rozdziale 1.2 proces testowy jest opisywany jako część składowa pełnego cyklu życia oprogramowania. Temat ten wynika z podstawowych koncepcji wprowadzonych w syllabusie dla poziomu podstawowego i zwraca szczególną uwagę na powiązanie procesu testowego z modelami wytwarzania oprogramowania i innymi procesami IT.

Systemy mogą przyjmować różnorodne formy, które mogą znacząco wpływać na wybór podejścia do testów. W rozdziale 1.3 wprowadzone zostały dwa specyficzne typy systemów, z którymi każdy tester musi być zaznajomiony: systemy systemów (czasami zwane mutisystemami, systemami złożonymi z innych systemów, ang. *multi-systems*) i systemy krytyczne ze względu na bezpieczeństwo.

Zaawansowani testerzy przy wprowadzaniu różnych aspektów testowania opisanych w tym syllabusie mierzą się z wieloma wyzwaniami w środowisku swojej organizacji, zespołu i zadań.

1.2 Testowanie w cyklu życia oprogramowania

Testowanie jest integralną częścią różnych modeli życia oprogramowania, takich jak:

- Model sekwencyjny (model kaskadowy, model V, model W)
- Model iteracyjny (Rapid Application Development RAD, model spiralny)
- Model przyrostowy (ewolucyjny i metodyki zwinne – ang. *Agile)*

Długofalowe podejście do testowania w cyklu życia oprogramowania powinno być uwzględnione w strategii testowej, która określa organizację testów, definicje procesów oraz wybór narzędzi i metod.

Procesy testowe nie powinny być wyizolowane, lecz powiązane z innymi procesami, takimi jak np.

- inżynieria i zarządzanie wymaganiami
- zarządzanie projektem

- zarządzanie konfiguracją i zmianą
- wytwarzanie oprogramowania
- utrzymanie (pielęgnacja) oprogramowania
- wsparcie techniczne
- tworzenie dokumentacji technicznej

W modelach sekwencyjnych wczesne planowanie testów oraz ich późniejsze wykonanie są ze sobą powiązane. Zadania testowe mogą na siebie nachodzić, bądź też być wykonywane równolegle.

Zarządzanie zmianą i zarządzanie konfiguracją są ważnymi procesami wspomagającymi testowanie oprogramowania. Bez poprawnego zarządzania zmianą nie można dokonać poprawnej oceny wpływu zmian na system. Bez poprawnego zarządzania konfiguracją równoległy rozwój różnych wersji oprogramowania może nie być możliwy (uprzednia praca może zostać utracona lub być źle zarządzana).

W zależności od potrzeb projektu, mogą być zdefiniowane dodatkowe poziomy testowania (w stosunku do tych określonych na poziomie podstawowym), jak np.:

- testy integracji sprzętu i oprogramowania
- testy integracji systemów
- testy integracji funkcjonalności
- testy integracji z innymi produktami w środowisku klienta

Każdy poziom testów powinien mieć określone:

- cele
- zakres
- śledzenie powiązań z podstawą testów
- kryteria wejścia i wyjścia (zakończenia testów)
- produkty testów włączając raporty
- techniki testowania
- miary i metryki
- narzędzia testowe
- zgodność ze standardami (organizacyjnymi lub ogólnymi)

W zależności od kontekstu, celów i zakresu każdy poziom testów może być rozważany odrębnie lub na poziomie projektu (np. by uniknąć niepotrzebnego powtarzania podobnego testu na różnych poziomach).

Czynności testowe muszą być zgodne z wybranym modelem życia oprogramowania: sekwencyjnym (np. kaskadowy, model V, model W), iteracyjnym (np. RAD lub model spiralny), lub przyrostowym (metodyki ewolucyjne lub zwinne – ang. *Agile*).

Na przykład w modelu V, podstawowy proces testowy ISTQB ® zastosowany do testów systemowych może wyglądać następująco:

- Planowanie testów systemowych rozpoczyna się równolegle z planowaniem projektu. Kontrola testów trwa aż do wykonania wszystkich testów systemowych i zamknięcia tej fazy testów.
- Analiza i projektowanie testów systemowych rozpoczyna się równolegle ze specyfikowaniem wymagań, budową projektu systemu i architektury (wysokopoziomowej) i specyfikacją modułów (niskopoziomową).
- Tworzenie środowiska do testów systemowych można rozpocząć podczas projektowania systemu, aczkolwiek znaczną część prac wykonuje się równolegle z kodowaniem i testowaniem modułów oraz innymi czynnościami implementacyjnymi, co często oznacza, że nad środowiskiem testowym pracujemy aż do dnia rozpoczęcia testów systemowych.
- Wykonanie testów systemowych rozpoczyna się, gdy spełnione są kryteria wejściowe (lub świadomie z nich rezygnujemy), co na ogół oznacza, że wykonane są przynajmniej testy modułowe i testy integracyjne modułów. Testy wykonuje się aż do spełnienia kryteriów zakończenia testów.
- Ocena kryteriów zakończenia testów systemowych i raportowanie ich wyników odbywa się przez cały czas wykonywania testów systemowych, z coraz większą częstotliwością i pilnością, w miarę zbliżania się terminu zakończenia projektu.
- Czynności zamykające testy systemowe rozpoczynają się po spełnieniu kryteriów zakończenia testów. Niekiedy czeka się na zakończenie testów akceptacyjnych i wszystkich czynności związanych z zamknięciem projektu.

Kierownik testów, jeszcze na etapie planowania testów, bądź projektu, powinien zaplanować sposób powiązania procesu testowania z modelem wytwarzania oprogramowania. Dla bardzo złożonych projektów, jak tworzenie systemu systemów (często tworzonych na potrzeby wojska lub wielkich korporacji) procesy testowe muszą być nie tylko powiązane z modelem wytwarzania oprogramowania, ale także dostosowywane do potrzeb konkretnego projektu (np. gdy łatwiej znaleźć defekt na wysokim poziomie testów, niż na niskim).

1.3 Systemy specyficzne

1.3.1 Systemy systemów (ang. *system of systems*)

Systemy systemów to zbiór współpracujących systemów (włączając w to sprzęt, pojedyncze aplikacje oraz komunikację), powiązanych ze sobą, aby osiągnąć wspólny cel, bez pojedynczej struktury zarządzającej. Właściwości i ryzyka związane z systemami systemów wymieniono poniżej:

- Stopniowe łączenie niezależnych systemów, tak by uniknąć tworzenia całego systemu od podstaw. Można to osiągnąć na przykład poprzez integrację zakupionych systemów „z półki” (ang. *COTS*) z ograniczoną ilością dodatkowego oprogramowania.
- Technologiczna i organizacyjna złożoność (np. zależności pomiędzy różnymi interesariuszami) stwarza ryzyko dla efektywnego zarządzania. Różne modele wytwarzania oprogramowania stosowane do integrowanych podsystemów mogą doprowadzić do problemów komunikacyjnych pomiędzy zaangażowanymi zespołami (programiści, testerzy, dostawcy rozwiązań, wdrożeniowcy, użytkownicy itp.). Całościowe zarządzanie systemem systemów musi adresować nieodłączną techniczną złożoność zadania, jakim jest łączenie różnych (pod)systemów, i wspomagać rozwiązywanie pojawiających się kwestii organizacyjnych,

takich jak wykonywanie pracy przez pracowników spoza firmy (ang. *outsourcing*) lub nawet spoza kraju (ang. *offshoring*).
- Poufność i ochrona specyficznej wiedzy (ang. *know-how*), komunikacja między organizacjami (np. sektor państwowy i prywatny), jak również decyzje prawne (np. zapobieganie praktykom monopolistycznym) mogą powodować, że system musi być traktowany jako system systemów.
- Systemy systemów są ze swej natury mniej niezawodne niż pojedyncze systemy, ponieważ jakiekolwiek ograniczenie z jednego (pod)systemu stosuje się automatycznie do całego rozwiązania.
- Wysoki poziom technicznego i funkcjonalnego współdziałania wymaganego w systemie systemów powoduje, że testy integracyjne stają się niezmiernie ważne i wymagają dobrze określonych i uzgodnionych interfejsów.

1.3.1.1 Zarządzanie i testowanie systemu systemów

Budowa systemu systemów jest wyzwaniem dla zarządzania projektem i procesu zarządzania konfiguracją poszczególnych modułów. Specyfika tych systemów jednocześnie znacząco wpływa na zadania z zakresu zapewnienia jakości i zdefiniowane procesy. Z systemami systemów powiązany jest formalny cykl wytwarzania, kamienie milowe i przeglądy.

1.3.1.2 Charakterystyka cyklu życia systemu systemów

Każdy poziom testowania systemu systemów ma następujące dodatkowe charakterystyki, oprócz tych opisanych w rozdziale 1.2:

- Wielokrotne poziomy integracji i zarządzania wersjami
- Długi czas trwania projektu
- Formalny obieg informacji pomiędzy członkami projektu
- Niewspółbieżny rozwój (pod)systemów i wymagań dla testów regresji na poziomie systemu systemów
- Testowanie pielęgnacyjne spowodowane wymianą pojedynczego modułu w wyniku starzenia się lub modyfikacji

W przypadku systemu systemów, poziom testowania musi być rozpatrywany na odpowiednim poziomie szczegółowości i na różnych poziomach integracji. Na przykład „poziom testów systemowych" dla jednego z (pod)systemów może być rozpatrywany jako „poziom testów modułowych" całego rozwiązania.

Na ogół każdy indywidualny system (składający się na całość rozwiązania) testów przechodzi przez wszystkie poziomy testowania, a po zintegrowaniu z resztą (pod)systemów jest dodatkowo testowany.

Specyficzne aspekty zarządzania systemami złożonymi z innych systemów zostały opisane w rozdziale 3.11.2.

1.3.2 Systemy krytyczne ze względu na bezpieczeństwo

Systemy krytyczne ze względu na bezpieczeństwo to takie systemy, których przerwa lub ograniczenie w funkcjonowaniu (np. spowodowane nieprawidłową operacją lub przez nieuwagę) mogą spowodować katastrofalne lub krytyczne konsekwencje. Dostawca systemu krytycznego ze względu na bezpieczeństwo może być odpowiedzialny za zniszczenia lub wypłacenie odszkodowań, a czynności testowe są wykonywane także by

zmniejszyć tę odpowiedzialność. Testowanie dostarcza dowodów, że system był adekwatnie testowany, by zapobiec katastrofie lub innym konsekwencjom krytycznym.

Przykłady systemów krytycznych ze względu na bezpieczeństwo obejmują systemy kontroli lotów, systemy transakcyjne, systemy sterowania elektrownią jądrową, systemy medyczne i inne.

W systemach krytycznych ze względu na bezpieczeństwo powinny być uwzględniane następujące aspekty:

- śledzenie powiązań pomiędzy wymaganiami a regulacjami i zarządzanie dowodami zgodności z regulacjami
- rygorystyczne podejście do wytwarzania i testowania
- analiza bezpieczeństwa
- redundantna architektura i jej ograniczenia
- zorientowanie na jakość
- wysoka jakość dokumentacji (szczegółowość i kompleksowość dokumentacji)
- wysoka podatnośc na audyty

Specyficzne aspekty zarządzania systemami krytycznymi ze względu na bezpieczeństwo zostały opisane w rozdziale 3.11.3.

1.3.2.1 Zgodność z regulacjami

Systemy krytyczne ze względu na bezpieczeństwo są często przedmiotem rządowych, międzynarodowych lub branżowych uregulowań lub standardów (patrz także 8.2 Uznane standardy). Mogą one odnosić się do procesu wytwórczego i struktury organizacyjnej lub do wytwarzanego produktu. Chcąc wykazać zgodność struktury organizacyjnej i procesu wytwórczego z regulacjami, można posłużyć się audytami lub schematami organizacyjnymi.

Chcąc wykazać zgodność produktu ze specyficznymi regulacjami, należy udowodnić, że każde wymaganie w tej regulacji jest odpowiednio zaadresowane. W takich przypadkach, konieczne jest pełne śledzenie pokrycia wymagań (od wymagania do dowodów jego zaadresowania). To wpływa na zarządzanie, cykl wytwarzania, testowanie i wybór procesów certyfikacji, dokonywanej przez uznane organy, przez cały czas wytwarzania.

1.3.2.2 Systemy krytyczne ze względu na bezpieczeństwo i złożoność

Wiele złożonych systemów i systemów systemów ma moduły krytyczne ze względu na bezpieczeństwo. Czasami aspekt bezpieczeństwa nie jest widoczny na poziomie pojedynczego systemu, lecz dopiero na poziomie całego rozwiązania (na przykład awionika w samolotach, systemy zarządzania ruchem samolotów).

Przykład: router nie jest sam w sobie krytycznym systemem, ale może się nim stać, gdy wymagają go krytyczne funkcjonalności np. w obsłudze telemedycznej.

Zarządzanie ryzykiem, które redukuje prawdopodobieństwo i/lub wpływ ryzyka jest podstawą dla wytwarzania i testowania systemów krytycznych ze względu na bezpieczeństwo (patrz rozdział 3). Dodatkowo powszechnie stosowana w tym kontekście jest analiza przyczyn i skutków awarii (FMEA) (patrz rozdział 3.10) i analiza wspólnych powodów awarii oprogramowania (ang. *Software Common Cause Failure Analysis*) .

1.4 Metryki i miary

Podczas cyklu życia oprogramowania (np. podczas planowania, śledzenia pokrycia, oceny nakładu pracy itp.) powinny być stosowane różne metryki (liczby) i miary (trendy, grafy itp.). W każdym przypadku musi być zdefiniowana podstawa (wartość początkowa), a sam postęp śledzony z uwzględnieniem relacji z tą wartością.

Miary i metryki mogą dotyczyć:

- planowanego harmonogramu, postępu prac i zmian w czasie
- wymagań, ich zmian i wpływu na harmonogram, zasoby i zadania
- nakładu pracy (pracochłonności), użycia zasobów i ich zmian w czasie
- kamieni milowych i zakresu, oraz ich zmian w czasie
- kosztów bieżących i estymowanych do zakończenia zadań
- ryzyk i czynności im zapobiegających oraz ich zmian w czasie
- defektów wykrytych, usuniętych, czasu naprawy

Użycie metryk umożliwia testerom raportowanie danych przełożonym w sposób logiczny i konsekwentny, oraz śledzenie postępu prac w czasie.

Należy wziąć pod uwagę trzy obszary :

- Definicja metryk: powinien być zdefiniowany ograniczony zbiór użytecznych metryk. Po tym, kiedy metryki zostały już zdefiniowane, należy uzgodnić ich jednoznaczną interpretację ze wszystkimi interesariuszami, tak aby uniknąć dyskusji w przyszłości, gdy wartości metryki będą się zmieniać. Metryki winny być definiowane w oparciu o cele procesu lub zadania, modułu lub systemu, osoby lub zespołu. Często pojawia się tendencja do definiowania zbyt wielu metryk, zamiast tylko tych najbardziej odpowiednich.
- Śledzenie metryk: raportowanie i łączenie metryk powinno być jak najbardziej zautomatyzowane, by zredukować czas potrzebny na wyprodukowanie nieprzetworzonych („surowych") danych. Różnorodność danych w czasie dla konkretnej metryki może odzwierciedlać inne informacje, niż interpretacje uzgodnione w fazie definicji.
- Raportowanie metryk: celem jest dostarczenie natychmiastowego zrozumienia informacji, do celów zarządczych. Prezentacja może pokazywać wartość metryki w konkretnym czasie lub zmianę wartości metryk w funkcji czasu, tak by można było określić trendy.

1.5 Etyka

Zaangażowanie się w testowanie umożliwia dostęp do informacji zastrzeżonej lub uprzywilejowanej. Kodeks etyczny jest niezbędny, także dlatego, żeby zapewnić, że informacje nie będą używane w sposób nieprawidłowy. Po zapoznaniu się z kodeksami etyki dla inżynierów z ACM i IEEE, ISTQB® ustanawia następujący kodeks etyczny:

SPOŁECZEŃSTWO – certyfikowani testerzy oprogramowania powinni postępować zgodnie z interesem społecznym.

KLIENT I PRACODAWCA – certyfikowani testerzy oprogramowania powinni postępować w jak najlepiej pojętym interesie ich klientów i pracodawców, zgodnie z interesem społecznym.

PRODUKT – certyfikowani testerzy oprogramowania powinni dostarczać produkty najwyższej jakości.

OSĄD – certyfikowani testerzy oprogramowania będą zachowywać uczciwość i niezależność w swych profesjonalnych osądach.

ZARZĄDZANIE – certyfikowani kierownicy i liderzy testów będą popierać i promować etyczne zachowanie w testowaniu oprogramowania.

ZAWÓD – certyfikowani testerzy oprogramowania będą promować uczciwość i umacnianie reputacji swego zawodu, zgodnie z interesem społecznym.

KOLEŻEŃSTWO – certyfikowani testerzy oprogramowania będą zachowywać się uczciwie i pomocnie w stosunku do swoich kolegów; będą promować współpracę z programistami.

WŁASNE – certyfikowani testerzy oprogramowania będą przez całe życie dokształcać się w swym zawodzie i będą promować etyczne zachowanie w praktyce zawodowej.

2. Procesy testowe

Terminologia:

BS 7925-2, kryteria wyjścia, IEEE 829, przypadek testowy, czynności zamykające testy, warunek testowy, kontrola testu, projektowanie testu, wykonanie testu, implementacja testów, planowanie testów, procedura testowa, skrypt testowy, raport podsumowujący testy, log (dziennik) testów

2.1 Wprowadzenie

W syllabusie poziomu podstawowego został opisany podstawowy proces testowy, na który składają się następujące czynności:

- planowanie i kontrola
- analiza i projektowanie
- implementacja i wykonanie
- ocena kryteriów zakończenia i raportowanie
- czynności zamykające testowanie

Chociaż powyższe czynności mogą być wykonywane sekwencyjnie, część z nich można zrównoleglić, np. analizę i projektowanie testów można przeprowadzać podczas implementacji i wykonania testów.

Jako że zarządzanie testami jest zasadniczo związane z procesem testowania, kierownicy testów powinni potrafić zastosować wszystkie elementy pokazane w tym rozdziale w odniesieniu do zarządzania konkretnym projektem. Analitykom testów i technicznym analitykom testów w większości przypadków wystarczy wiedza wymagana na poziomie podstawowym, za wyjątkiem jedynie zadań implementacji testów wymienionych powyżej. Wiedza wymagana do wykonywania tych zadań jest przedstawiona w niniejszym rozdziale w sposób ogólny, w szczegółowy sposób zastosowana zostanie w rozdziale 4 „Techniki testowania" oraz w rozdziale 5 „Testowanie właściwości oprogramowania".

2.2 Modele procesu testowania

Modele procesowe mają charakter jedynie abstrakcyjny i przybliżony. Modele procesu testowania nie opisują całej złożoności, niuansów i czynności, które składają się na rzeczywiste projekty lub przedsięwzięcia. Powinny być postrzegane jako pomoc w zrozumieniu i organizacji procesu testowania, nie zaś jako jedyny słuszny sposób jego opisu. Niniejszy syllabus używa jako przykładu procesu opisanego w syllabusie poziomu podstawowego (patrz wyżej), niemniej jednak istnieją również inne modele procesu testowego; trzy przykłady takich modeli zostały przedstawione poniżej. Wszystkie one są równocześnie modelami procesu testowania oraz modelami usprawniania procesu testowania (*Practical Software Testing* zawiera *Test Maturity Model*) i na podstawie przyjętych kryteriów - dzielą proces testowania na poziomy dojrzałości. Wspomniane trzy modele procesu testowania, jak również TPI®, są omówione w podrozdziale 8.3 Usprawnianie procesu testowania.

- *Practical Software Testing – Test Maturity Model* [Burnstein03]

- Krytyczne Procesy Testowania (ang. *Critical Testing Processes CTP* [Black03])
- Proces Systematycznego Testowania i Oceny (ang. *Systematic Test and Evaluation Process* – STEP)

2.3 Planowanie i kontrola testów

W tym podrozdziale skupiono się na procesach planowania i kontroli testów.

Większa część planowania jest wykonywana w fazie początkowej testowania. Składa się z rozpoznania i wdrożenia wszystkich czynności oraz zasobów wymaganych do wypełnienia misji testów i osiągnięcia celów zdefiniowanych w strategii testowania.

Testowanie oparte na ryzyku (patrz rozdział 3 „Zarządzanie testowaniem") jest stosowane w celu przekazania do procesu planowania testów informacji o czynnościach potrzebnych do zmniejszenia zidentyfikowanego ryzyka produktowego. Na przykład, jeżeli zostało stwierdzone, że w specyfikacji projektowej zwykle znajdowane są poważne usterki, w procesie planowania testów można uwzględnić dodatkowo testy statyczne specyfikacji (przeglądy), jeszcze zanim rozpoczęte zostanie kodowanie. Testowanie oparte na ryzyku dostarcza również informacji o priorytetach zadań testowych.

Pomiędzy przedmiotem testowania, warunkami testowymi, przypadkami testowymi oraz procedurami testowymi mogą istnieć złożone relacje, takie jak wiele do wielu. Muszą one zostać zrozumiane, aby planowanie i kontrola testów mogły być skutecznie wdrożone.

Kontrola testów jest aktywnością ciągłą. Polega na porównywaniu rzeczywistego postępu prac z planem i raportowaniu statusu, włączając w to odchylenia od planu. Kontrola testów umożliwia realizację spełnienia misji, strategii oraz celów testów, łącznie z rewizją czynności planowania testów w razie potrzeby.

Kontrola testów musi reagować na informacje generowane przez testowanie, jak również na zmieniające się warunki, w których prowadzony jest projekt lub przedsięwzięcie. Na przykład, jeżeli testowanie dynamiczne ujawni skumulowanie błędów w obszarach, które zostały ocenione jako mało podatne na usterki, lub jeżeli czas wykonania testów zostanie skrócony z powodu ich opóźnionego rozpoczęcia, analiza ryzyka i plan muszą zostać skorygowane. Może to spowodować zmianę priorytetów testów lub realokację zasobów w kontekście testów pozostałych do wykonania.

Zawartość dokumentów planowania testów została omówiona w rozdziale 3 „Zarządzanie testowaniem".

Metryki wspomagające planowanie i kontrolę testów mogą zawierać:

- pokrycie testów oraz ryzyka
- informacje o usterkach i ich wykrywaniu
- planowaną i rzeczywistą pracochłonność przygotowania testaliów i wykonania przypadków testowych

2.4 Analiza i projektowanie testów

Podczas planowania testów zostaną zidentyfikowane cele testowania. Proces analizy i projektowania testów korzysta z tych celów do:

- zidentyfikowania warunków testowych
- utworzenia przypadków testowych, które sprawdzają owe warunki testowe

Kryteria priorytetyzacji określone podczas analizy ryzyka oraz planowania testów powinny być wykorzystywane w całym procesie testowania, od analizy i projektowania do implementacji i wykonania testów.

2.4.1 Identyfikacja warunków testowych

Warunki testowe są identyfikowane poprzez analizę podstaw i celów testów, w celu stwierdzenia, co należy przetestować używając technik testowych określonych w Strategii Testów i/lub Planie Testów.

Decyzja o określeniu poziomu i struktury warunków testowych może być oparta na funkcjonalnych i niefunkcjonalnych cechach elementów testów, przy użyciu następujących wskazówek:

1. Granulacja podstawy testowania: np. wymagania wysokiego poziomu mogą być podstawą do określenia warunków testowych wysokiego poziomu np. „Udowodnij, że fortmatka X działa”, z którego może zostać wyprowadzony warunek niższego poziomu „Udowodnij, że formatka X odrzuca numer konta, którego długość jest mniejsza o jeden znak od poprawnej długości”.

2. Elementy ryzyka produktu: np. dla funkcji i cech wysokiego ryzyka celem może być zdefiniowanie szczegółowych warunków testowych niskiego poziomu.

3. Wymagania dotyczące raportowania dla kierownictwa oraz możliwość śledzenia powiązań pomiędzy informacjami.

4. Decyzja o nie implementowaniu przypadków testowych, a zamiast tego pracy tylko z warunkami testowymi np. przez użycie warunków testowych do sterowania nieformalnym testowaniem.

2.4.2 Tworzenie przypadków testowych

Przypadki testowe projektuje się krok po kroku poprzez opracowanie i udoskonalanie zidentyfikowanych warunków testowych, przy użyciu technik projektowania testów (patrz rozdział 4) określonych w Strategii Testów. Przypadki testowe powinny być powtarzalne, weryfikowalne oraz zawierać informację o powiązaniu z wymaganiami, na podstawie których powstały.

Projektowanie przypadków testowych oparte jest o:

- warunki wstępne, takie jak wymagania na środowisko testowe (projektowe lub lokalne) i plan jego dostarczenia
- wymagania na dane testowe
- oczekiwane wyniki i warunki końcowe.

Szczególnym wyzwaniem często jest określenie oczekiwanych wyników testu, to jest identyfikacja co najmniej jednej wyroczni testowej, która może zostać użyta w teście. Podczas określania oczekiwanych wyników, testerzy biorą pod uwagę nie tylko wyniki wyświetlane na ekranie, ale też warunki końcowe dotyczące danych oraz stan środowiska.

Czynności te teoretycznie mogą być proste, jeśli podstawa testów jest dobrze zdefiniowana. Jednakże podstawy testów są często niezrozumiałe, niespójne, brakuje im pokrycia kluczowych obszarów lub po prostu nie istnieją. W takich przypadkach tester musi mieć dostęp do wiedzy dziedzinowej. Nawet jeżeli podstawa testów jest dobrze zdefiniowana, złożone interakcje skomplikowanych wywołań i odpowiedzi mogą spowodować, że definicja oczekiwanych wyników będzie trudna, dlatego wyrocznia testowa staje się tu niezbędna.

Wykonywanie testów bez możliwości określenia poprawności ich rezultatów ma bardzo małą wartość dodaną, powoduje generowanie nieprawidłowych zgłoszeń incydentów oraz buduje fałszywe zaufanie do systemu.

Czynności opisane powyżej mają zastosowanie do wszystkich poziomów testowania, chociaż należy pamiętać, że zależnie od poziomu podstawy testu mogą się różnić. Na przykład testy akceptacyjne wykonywane przez użytkownika mogą być oparte głównie o specyfikację wymagań, przypadki użycia i procesy biznesowe, podczas gdy testy modułowe mogą zostać oparte na projekcie niskiego poziomu.

Podczas tworzenia warunków testowych i przypadków testowych powstaje zbiór dokumentacji testowej. Standardem opisującym tę dokumentację jest norma IEEE 829. Opisuje ona główne typy dokumentów dotyczących analizy i projektowania testów (Projekt Testów oraz Specyfikację Przypadków Testowych), jak również implementacji testów.

W praktyce poziom udokumentowania produktów testowania może się bardzo różnić. Może on na przykład zależeć od:

- ryzyka projektowego (co może, albo nie musi zostać udokumentowane)
- wartości dodanej dokumentacji dla projektu
- stosowanych standardów
- zastosowanego modelu cyklu życia (np. w metodykach zwinnych - ang. Agile - dokonuje się próby zminimalizowania ilości dokumentacji, zastępując ją częstą komunikacją w zespole)
- wymagania dotyczącego możliwości śledzenia od podstaw testów, poprzez analizę i projektowanie testów.

W zależności od zakresu testów, w ramach analizy i projektowania testów można również zajmować się cechami jakościowymi przedmiotu testów. Standard ISO 9126 może tu służyć jako użyteczne odniesienie. Podczas testowania systemów sprzętowo-programowych mogą być wzięte pod uwagę dodatkowe charakterystyki.

Proces analizy i projektowania testów może być ulepszony przez przeglądy i analizę statyczną. Na przykład przeprowadzenie analizy testów i projektowania testów na podstawie specyfikacji wymagań

jest doskonałym sposobem na przygotowanie się do spotkania przeglądu wymagań. Podobnie, przeglądom i analizie statycznej powinny podlegać produkty testowe, takie jak testy, analiza ryzyka oraz plany testów.

Podczas projektowania testów mogą zostać zdefiniowane szczegółowe wymagania na infrastrukturę testową, chociaż w praktyce zdarza się, że zostają one domknięte dopiero podczas implementacji testów. Należy mieć na uwadze, iż infrastruktura testowa składa się nie tylko z przedmiotu testów i testaliów, lecz również np. pokoi i ich wyposażenia, kadry, oprogramowania, narzędzi, urządzeń zewnętrznych, sprzętu komunikacyjnego, uprawnień użytkownika oraz wszystkich innych elementów wymaganych do przeprowadzenia testów.

Przykładowe metryki pomocne w monitorowaniu analizy i projektowania testów:

- odsetek wymagań pokrytych przez warunki testowe
- odsetek warunków testowych pokrytych przez przypadki testowe
- liczba defektów znalezionych podczas analizy i projektowania testów

2.5 Implementacja i wykonanie testów

2.5.1 Implementacja testów

Implementacja testów polega na ułożeniu przypadków testowych w procedury/scenariusze testowe (skrypty testowe), doprecyzowywaniu danych testowych i środowisk testowych oraz stworzeniu harmonogramu wykonania testów, aby można było rozpocząć wykonanie przypadków testowych. W zakres implementacji testów wchodzi również sprawdzenie jawnych i niejawnych kryteriów wejścia dla danego poziomu testów.

Procedury testowe powinny być priorytetyzowane, aby zapewnić możliwie najefektywniejszą drogę osiągnięcia celów określonych w strategii np. w pierwszej kolejności wykonywać najważniejsze procedury testowe.

Na poziom szczegółowości i złożoność pracy wykonywanej podczas implementacji testów może wpływać szczegółowość produktów testowania (przypadków testowych i warunków testowych). W niektórych przypadkach może zachodzić potrzeba spełnienia wymagań prawnych i w takiej sytuacji testy powinny dostarczyć dowodów, że zostały spełnione odpowiednie standardy, takie jak np. United States Federal Aviation Administration's DO-178B/ED 12B.

Tak jak to zostało określone w podrozdziale 2.4, do testów potrzebne są dane testowe, a w niektórych przypadkach zestawy danych testowych mogą być całkiem duże. Podczas implementacji testów testerzy tworzą dane wejściowe i dane środowiskowe do załadowania do baz danych i innych podobnych repozytoriów. Testerzy odpowiadają również za przygotowanie skryptów i innych generatorów danych. Generatory przygotowują dane do obciążenia systemu podczas wykonywania testów.

Podczas implementacji testów, testerzy powinni ostatecznie ustalić i potwierdzić kolejność, wg której będą wykonywane testy manualne i automatyczne. W przypadku testów automatycznych, implementacja testów obejmuje również wytworzenie jarzma testowego i skryptów testowych. Testerzy powinni dokładnie sprawdzić wszystkie ograniczenia, które mogą wymusić wykonanie testów w konkretnej kolejności. Zależności w ramach środowiska testowego lub danych testowych muszą być znane i sprawdzone.

W ramach implementacji testów należy zająć się również środowiskiem lub środowiskami testowymi. Podczas tej fazy, a przed wykonaniem testów, powinno ono zostać w pełni skonfigurowane oraz sprawdzone. Niezwykle ważne jest dostosowanie środowiska testowego do celu testów. Środowisko testowe powinno umożliwiać wykrywanie błędów zgodnie z warunkami testowymi, działać prawidłowo, gdy nie występują awarie oraz - tam gdzie jest to potrzebne - dokładnie odzwierciedlać środowisko produkcyjne i środowisko użytkownika (np. dla wyższych poziomów testów).

Podczas implementacji testów, testerzy muszą upewnić się, że osoby odpowiedzialne za utworzenie i obsługę środowiska testowego są znane i dostępne, oraz że wszystkie testalia i narzędzia wspomagające testowanie, jak również powiązane procesy, są gotowe do pracy. Wlicza się w to zarządzanie konfiguracją, zarządzanie incydentami, logowanie testów i zarządzanie testami. Dodatkowo testerzy muszą sprawdzić procedury zbierania danych dla celów oceny kryteriów zakończenia i raportowania wyników testów.

Zalecane jest zrównoważone podejście do implementacji testów. Na przykład analityczne strategie testowania oparte na ryzyku są często mieszane ze strategiami dynamicznymi, wg których część testów odbywa się bez wcześniej przygotowanych skryptów.

Testowanie bez skryptów nie powinno być testowaniem ad hoc lub testowaniem pozbawionym celu, ponieważ takie testowanie może być nieprzewidywalne co do czasochłonności (testowanie w wytyczonych ramach czasowych - patrz SBTM). Na przestrzeni lat testerzy rozwinęli wiele technik opartych na doświadczeniu, takich jak ataki (patrz punkt 4.4 i [Whittaker03]), zgadywanie błędów [Myers79] oraz testowanie eksploracyjne. Nadal występuje w nich analiza testów, projektowanie testów oraz implementacja testów, ale głównie podczas wykonywania testów. W przypadku dynamicznych strategii testowania rezultaty każdego testu wpływają na analizę, projekt i implementację kolejnych testów. Mimo że strategie te są „lekkie” (ang. *leightweigth*) i często skuteczne w znajdowaniu błędów, wymagają one zaangażowania testerów - ekspertów, a czas ich trwania może być nieprzewidywalny. Techniki te często nie dają wystarczających informacji na temat pokrycia testów, a powtórzenie testów może być niemożliwe bez specjalnych narzędzi do testów regresywnych.

2.5.2 Wykonanie testów

Wykonanie testów rozpoczyna się z chwilą, gdy przedmiot testów zostaje dostarczony i spełnione zostają kryteria wejścia do wykonania testów. Testy powinny być wykonywane według procedur testowych, niemniej jednak należy zostawić testerowi trochę swobody, żeby zapewnić pokrycie dodatkowych, interesujących scenariuszy testowych i obserwację zachowania systemu podczas

testów (opis każdej awarii wykrytej podczas takich testów powinien zawierać również opis odstępstw od spisanej procedury testowej, który jest niezbędny do odtworzenia tej awarii). Testy automatyczne wykonywane są zgodnie z określonymi instrukcjami, bez żadnych odstępstw.

Esencją wykonywania testów jest porównanie wyników rzeczywistych z wynikami oczekiwanymi. Tę czynność tester musi wykonywać z wielką uwagą i skupieniem, w przeciwnym wypadku wysiłek projektowania i implementacji testów może zostać zmarnowany, np. awarie pozostaną niezauważone (wyniki pozornie pozytywne) albo poprawne zachowanie zostanie zakwalifikowane jako błędne (wyniki pozornie negatywne). Jeżeli wyniki rzeczywiste i oczekiwane się nie zgadzają, mówimy o incydencie. Incydentom należy się dokładnie przyjrzeć, żeby ustalić ich przyczynę (która może, ale nie musi być usterką w przedmiocie testów) a także, by zebrać dane pomocne w ich rozwiązaniu. Zarządzanie incydentami jest omówione w rozdziale 7.

Kiedy defekt zostanie zidentyfikowany, należy dokładnie przeanalizować specyfikację testów, aby upewnić się, że jest ona poprawna. Specyfikacja testów może być nieprawidłowa z wielu powodów, włączając w to problemy z danymi testowymi, defekty w dokumentach lub pomyłki podczas wykonania testów. Jeżeli specyfikacja testów okaże się niepoprawna, powinna zostać zaktualizowana, a test powinien zostać wykonany ponownie. Ponieważ zmiany w podstawie i przedmiocie testów mogą spowodować dezaktualizację specyfikacji testów, nawet gdy testy zostały pomyślnie wykonane wiele razy, testerzy powinni być świadomi tego, że zaobserwowane wyniki mogły zostać spowodowane niepoprawnymi testami.

Wyniki wykonania testów muszą być należycie zapisywane. Testy, które były wykonane, ale których rezultaty nie zostały zanotowane, mogą wymagać powtórnego wykonania w celu zidentyfikowania poprawnych rezultatów, co prowadzi do spadku efektywności i opóźnień. Należy zauważyć, że właściwe logowanie wyników może rozwiązać problemy ze śledzeniem pokrycia testów i ich powtarzalnością, które są typowe dla dynamicznych strategii testowania. Ze względu na fakt, iż przedmiot testów, testalia i środowiska testowe mogą się zmieniać, wyniki wykonania testów należy rejestrować w powiązaniu z testowaną wersją przedmiotu testów.

Log (dziennik) testowy zawiera chronologiczny zapis istotnych szczegółów wykonania testów.

Zapis wyników odnosi się zarówno do indywidualnych testów, jak i do zdarzeń. Każdy test powinien być jednoznacznie identyfikowalny, a jego status zapisany w trakcie wykonywania procedury testowej. Wszystkie wydarzenia mające wpływ na wykonanie testów powinny zostać zarejestrowane. Rejestracji powinna podlegać również informacja niezbędna do zmierzenia pokrycia testowego oraz udokumentowania powodów opóźnień i przerw w testach. Dodatkowo zapisywana powinna być informacja wspomagająca kontrolę testów, raportowanie postępu testów, pomiar kryteriów wyjścia i ulepszanie procesu testowania.

Logowanie może się różnić zależnie od poziomu testów i strategii testów. Na przykład, w przypadku wykonywania automatycznego testowania modułowego, automaty testowe zbierają większość informacji. Jeżeli wykonywane są ręczne testy akceptacyjne, kierownik testów może przygotować lub

łączyć logi testowe. W niektórych przypadkach, tak jak to było w implementacji testów, na logowanie mogą wpływać regulacje prawne lub wymagania audytowe.

Standard IEEE 829 zawiera opis informacji, które powinny znaleźć się w logu testowym:

- identyfikator logu testowego
- opis
- wpisy dotyczące aktywności i zdarzeń

Standard BS 7925-2 również zawiera opis informacji, która powinna być logowana.

W niektórych przypadkach użytkownicy lub klienci mogą uczestniczyć w wykonywaniu testów. Może to posłużyć jako metoda zdobywania ich zaufania do systemu, chociaż takie podejście zakłada, że w testach raczej nie wykrywa się błędów. Takie założenie jest często nieprawdziwe na wczesnych poziomach testów, ale może być poprawne podczas testów akceptacyjnych.

Przykładowe metryki służące do monitorowania implementacji i wykonania testów:

- odsetek skonfigurowanych środowisk testowych
- odsetek załadowanych rekordów danych testowych
- odsetek wykonanych warunków i przypadków testowych
- odsetek zautomatyzowanych przypadków testowych

2.6 Ocena spełnienia kryteriów wyjścia oraz raportowanie

Dokumentowanie i raportowanie dla celów monitorowania i kontroli postępu testów jest szczegółowo omówione w podrozdziale 3.6. Z poziomu procesu testowania, monitorowanie postępu testów pociąga za sobą zbieranie informacji potrzebnych do spełnienia wymagań dotyczących raportowania. Ta czynność zawiera również pomiar postępów prac.

Metryki wspomagające monitorowanie postępu testów i podjęcie decyzji o ich zakończeniu będą zawierać mapowanie na uzgodnione w fazie planowania testów kryteria wyjścia. Przykładowe metryki podano poniżej:

- liczba zaplanowanych i wykonanych warunków testowych, przypadków testowych lub specyfikacji testowych, z podziałem na te, które udało się wykonać bez usterek i te zakończone usterkami
- wszystkie znalezione (poprawione i ciągle otwarte) usterki z podziałem na wagi i priorytety
- liczba zmian (żądań zmian) zgłoszonych, zaakceptowanych (wykonanych) i przetestowanych
- koszty planowane w porównaniu do kosztów rzeczywistych
- planowany czas trwania w porównaniu do rzeczywistego czasu trwania
- zidentyfikowane obszary ryzyka z podziałem na te zaadresowane przez testy i pozostałe
- odsetek czasu testowania straconego w wyniku zdarzeń blokujących
- elementy powtórnie testowane
- całkowity planowany czas na testy w stosunku do czasu rzeczywiście przeznaczonego na testy

Dla raportowania testów norma IEEE 829 wyszczególnia Sumaryczny Raport z Testów, który składa się z następujących elementów:

- identyfikator raportu
- podsumowanie
- odstępstwa
- ogólna ocena
- podsumowanie wyników
- ocena
- podsumowanie czynności
- akceptacje

Raport z testów może zostać stworzony po zakończeniu każdego z poziomów testów, a także, gdy całe testy zostaną zakończone.

2.7 Czynności na zakończenie testowania

Kiedy wykonanie testów zostanie uznane za skończone, należy uchwycić kluczowe rezultaty i albo przekazać je odpowiedniej osobie, albo zarchiwizować. Łącznie nazywamy to czynnościami zamykającymi testy. Czynności zamykające testy można podzielić na cztery grupy:

1. Upewnienie się, że zadania testowe są rzeczywiście zakończone. Na przykład, wszystkie planowane testy powinny zostać wykonane lub świadomie pominięte, wszystkie znalezione defekty powinny być poprawione i zretestowane, odroczone do testów w kolejnych wersjach lub zaakceptowane jako stałe ograniczenia.
2. Dostarczenie wartościowych produktów pracy tym, którzy ich potrzebują. Na przykład defekty odroczone lub zaakceptowane powinny zostać zakomunikowane tym, którzy będą używali systemu lub wspomagali jego użycie. Testy i środowiska testowe powinny zostać przekazane tym, którzy będą wykonywali testy pielęgnacyjne. Kolejnym produktem może być zestaw testów regresywnych (manualnych lub automatycznych).
3. Organizacja i uczestnictwo w spotkaniach podsumowujących, wyciąganie wniosków na przyszłość (ang. *lessons learned*), gdzie ważne wnioski (zarówno z procesu testowania, jak i całego cyklu tworzenia oprogramowania) mogą zostać udokumentowane, i gdzie mogą powstać plany zapewnienia, że popełnione błędy nie zostaną powtórzone w przyszłości, a te których nie da się naprawić, będą uwzględniane w planach projektu. Na przykład:
    a. z powodu późnego wykrycia nieprzewidzianych skupisk defektów, zespół mógł dojść do wniosku, że w przyszłych projektach w sesje analizy ryzyka jakościowego powinni zostać włączeni dodatkowi użytkownicy
    b. szacowania mogły znacznie odbiegać od rzeczywistości, o czym należy pamiętać dokonując kolejnych estymacji, zwracając uwagę na przyczyny rozbieżności np. czy powodem było nieefektywne testowanie, czy zaniżone estymacje
    c. trendy oraz analiza przyczyn i skutków defektów, przez powiązanie ich wstecz z momentem, kiedy i dlaczego wystąpiły, wraz z poszukiwaniem zależności np. czy późne żądania zmian wpłynęły na jakość analizy i kodowania, albo poszukiwanie złych praktyk np. opuszczanie poziomów testów, dzięki których usterki mogłyby być znalezione wcześniej, a koszty zaoszczędzone (dla zaoszczędzenia czasu), trendy

usterek w powiązaniu z nowymi technologiami, zmianami w zespole projektowym lub brakiem wymaganych umiejętności
   d. identyfikacja potencjalnych udoskonaleń procesu
4. Archiwizacja wyników, logów, raportów i innych dokumentów i produktów w systemie zarządzania konfiguracją, w powiązaniu z testowanym systemem. Na przykład plan testów i plan projektu powinny zostać umieszczone w archiwum planowania z jasnym wskazaniem na system i wersję, dla której zostały stworzone.

Powyższe czynności, mimo iż bardzo ważne, są jednak często pomijane. Czynności, o których tutaj mowa powinny być jawnie wymienione w planie testów.

Częstą praktyką jest pomijanie tych zadań. Zwykle z powodu rozwiązania zespołu projektowego, nacisków na udostępnienie zasobów lub z powodu napiętego harmonogramu kolejnych projektów albo z powodu zawodowego wypalenia się ludzi. W projektach zlecanych w formie kontraktów takich jak rozwój oprogramowania na zamówienie, wymagane czynności kończące powinny zostać wyszczególnione w umowie.

Przykładowe metryki monitorujące czynności zamykające test:

- odsetek przypadków testowych wykonanych podczas wykonania testów (pokrycie)
- odsetek przypadków testowych umieszczonych w repozytorium przypadków testowych celem ich ponownego użycia
- stosunek testów zautomatyzowanych do tych, które miały być zautomatyzowane
- odsetek przypadków testowych uznanych za testy regresywne
- odsetek pozostałych błędów, które zostały zamknięte bez poprawienia (np. odłożone, odrzucone, żądania zmian)
- odsetek produktów, które zostały zidentyfikowane i zarchiwizowane.

3. Zarządzanie testowaniem

Terminologia

FMEA (analiza przyczyn i skutków awarii), jednopoziomowy plan testów, główny plan testów, ryzyko produktowe, ryzyko projektowe, testowanie na podstawie ryzyka, analiza ryzyka, identyfikacja ryzyka, poziom ryzyka, zarządzanie ryzykiem, łagodzenie ryzyka, rodzaj ryzyka, kontrola testów, zarządzanie testowaniem w sesjach, szacowanie testów, poziom testów, zarządzanie testami, monitorowanie testów, plan testów, polityka testów, analiza punktów testowych, harmonogramowanie testów, strategia testów, *wide band delphi*.

3.1 Wprowadzenie

Cały niniejszy rozdział dotyczy tematyki szczególnie niezbędnej dla kierowników testów.

3.2 Dokumentacja służąca do zarządzania testowaniem

Zwykle dokumentację tworzy się w ramach zarządzania testowaniem. Choć konkretne dokumenty różnie się nazywają, najczęściej spotykane rodzaje dokumentów służących do zarządzania testowaniem w projektach i w organizacjach to:

- Polityka testów, określająca podejście do testowania w danej organizacji (a niekiedy także politykę zapewnienia jakości).
- Strategia testów (albo podręcznik testowania), określająca metody testowania w organizacji, w tym zarządzanie ryzykiem produktowym i ryzykiem projektowym, podział testowania na kroki, poziomy lub fazy, oraz wysokopoziomowe czynności związane z testowaniem.
- Główny plan testów (albo plan testów w projekcie, lub podejście do testowania), opisujący sposób zastosowania strategii testów w określonym projekcie, w tym realizowane poziomy testów, oraz zależności między nimi.
- Jednopoziomowy plan testów (plan testów danego poziomu), opisujący czynności, które mają być zrealizowane na danym poziomie testów, uzupełniający główny plan testów.

W niektórych organizacjach i projektach, opisane wyżej rodzaje dokumentów bywają łączone w jeden dokument; zawartość tych dokumentów może się także znajdować w innych dokumentach. Może się również zdarzyć, że będzie to wiedza niepisana, postrzegana jako intuicyjna lub zaliczająca się do tradycyjnych metodyk testowania. W większych i bardziej sformalizowanych organizacjach i projektach dokumenty te traktowane są jako udokumentowane produkty pracy, natomiast w mniejszych i mniej sformalizowanych - występują one w mniejszej ilości.

Niniejszy syllabus opisuje każdy z tych dokumentów oddzielnie, choć w praktyce zastosowanie każdego typu określone jest przez kontekst organizacyjny i projektowy.

3.2.1 Polityka testów

Polityka testów opisuje filozofię/podejście stosowane przez organizację wobec testowania (a także – jeśli jest to możliwe - wobec zapewnienia jakości).

Polityka testów (spisana, bądź pośrednio zdefiniowana poprzez promowany kierunek zarządzania), ustanawia ogólne cele, jakie organizacja chce osiągnąć poprzez testowanie.

Politykę testów może sporządzić dział IT, dział badawczo-rozwojowy lub dział wytwarzania oprogramowania, najważniejsze jednak, aby odzwierciedlała ona promowane w organizacji wartości odnoszące się do testowania.

W niektórych przypadkach, polityka testów będzie uzupełnieniem lub fragmentem szerzej rozumianej polityki jakości, określającej ogólne wartości i cele zarządzania w kontekście jakości.

Jeśli polityka testów ma formę pisemną, jest to zwykle krótki, wysokopoziomowy dokument, który:

- podaje definicję testowania tj. budowanie zaufania do systemu (że działa, jak powinien) oraz znajdowanie defektów.
- określa podstawowy proces testowy, na przykład: planowanie oraz nadzorowanie testów, analiza i projektowanie testów, implementacja i wykonywanie testów, ocena kryteriów wyjścia oraz raportowanie wyników testów i zakończenie testów.
- opisuje, jak oceniać efektywność oraz skuteczność testowania, na przykład poprzez Odsetek Wykrytych Błędów (OWB) oraz względny koszt defektów wykrytych podczas testowania, w porównaniu do kosztu defektów wykrytych po wydaniu oprogramowania.
- definiuje pożądane cele jakości, takie jak niezawodność (na przykład mierzoną częstotliwością awarii) lub użyteczność.
- określa czynności służące udoskonalaniu procesu testowego, na przykład zastosowanie Modelu Dojrzałości Testów (TMM) lub Modelu Usprawniania Procesu Testowego (TPI), albo realizację zaleceń wynikających z analizy wcześniejszych projektów.

Polityka testów może odnosić się do czynności testowych zarówno w procesie wytwarzania, jak i podczas utrzymania oprogramowania. Może także odwoływać się do standardu definiującego terminologię związaną z testowaniem, która ma być stosowana w całej organizacji.

3.2.2 Strategia testów

Strategia testów opisuje metody testowania w organizacji, w tym zarządzanie ryzykiem produktowym i projektowym, podział testowania na poziomy lub fazy, oraz wysokopoziomowe czynności związane z testowaniem.

Strategia testów oraz opisane w niej procesy i czynności, powinny być zgodne z polityką testów oraz stanowić ogólne wymagania na testy dla organizacji, bądź dla jednego lub wielu projektów.

Jak zostało opisane w syllabusie poziomu podstawowego, strategie testów, zwane również podejściem do testowania, można podzielić w zależności od tego, kiedy zaczyna się projektowanie testów:

- Strategie prewencyjne, na bazie których testy projektowane są wcześnie a celem jest zapobieganie defektom.
- Strategie reaktywne, w których projektowanie testów rozpoczyna się po wytworzeniu oprogramowania lub systemu.

Do typowych strategii testów (lub podejść do testowania) zaliczamy:

- Strategie analityczne, takie jak testowanie na podstawie ryzyka.
- Strategie na podstawie modelu, takiego jak profil operacyjny.
- Strategie metodyczne – np. testowanie na podstawie atrybutów jakości.
- Strategie zgodne z procesem lub ze standardem – np. ze standardem IEEE 829.
- Strategie dynamiczne i heurystyczne, wykorzystujące np. użycie ataków na oprogramowanie.
- Strategie konsultatywne, takie jak testowanie z uwzględnieniem potrzeb użytkownika.
- Strategie testów regresywnych, takie jak szeroko zakrojona automatyzacja.

Strategie można łączyć. Wybrana strategia powinna być dostosowana do potrzeb i środków organizacji. Organizacje mogą dopasowywać strategie do poszczególnych działań oraz projektów.

Strategie często wyjaśniają ryzyka projektowe i produktowe oraz opisują, w jaki sposób zarządza się nimi w procesie testowania. W takim przypadku należy wprost wyjaśnić związki między ryzykami a testowaniem, tak samo jak sposoby zmniejszania i zarządzania tymi ryzykami.

Strategia testów może opisywać stosowane poziomy testów. W takim przypadku powinna określać ogólnie kryteria wejścia i zakończenia każdego poziomu oraz relacje między poziomami (na przykład podział celów pokrycia testowego).

Strategia testów może także określać co następuje:

- Procedury integracji
- Techniki specyfikacji (projektowania) testów
- Niezależność testowania (może być różna zależnie od poziomu testów)
- Obowiązkowe i opcjonalne standardy
- Środowiska testowe
- Automatyzację testów
- Ponowne użycie artefaktów procesu wytwarzania oprogramowania oraz procesu testowania
- Retesty oraz testy regresywne
- Nadzorowanie i raportowanie testów
- Pomiary i metryki testowe
- Zarządzanie incydentami
- Zarządzanie konfiguracją testaliów

Należy definiować zarówno krótkofalowe, jak i długofalowe strategie testów. Strategie mogą zostać opisane w jednym, bądź w kilku dokumentach. Dla różnych organizacji i projektów, odpowiednie są różne strategie. Przykładowo, przy wytwarzaniu aplikacji krytycznych ze względu na bezpieczeństwo lub na zabezpieczenia, może być właściwe zastosowanie bardziej rozbudowanej strategii niż w innych przypadkach.

3.2.3 Główny plan testów

Główny plan testów opisuje zastosowanie strategii testów do określonego projektu, w tym poziomy testów oraz ich relacje. Główny plan testów powinien być spójny z polityką i strategią testów, a tam gdzie od nich odbiega, niezgodności powinny być wyjaśnione. Główny plan testów powinien uzupełniać plan projektu oraz procedury operacyjne (ang. *operations guide*) o prace związane z testowaniem, a będące częścią tego projektu lub działania.

Choć szczegółowa zawartość głównego planu testów różni się w zależności od organizacji, standardów dokumentacji oraz stopnia sformalizowania projektu, to zwykle główny plan testów zawiera:

- Elementy testowane i te, które nie podlegają testowaniu
- Atrybuty jakościowe testowane i nie testowane
- Budżet i harmonogram testów (dopasowany do budżetu projektu lub budżetu operacyjnego)
- Cykle wykonywania testów oraz ich związek z planem wydawania wersji oprogramowania
- Biznesowe uzasadnienie wartości testowania
- Zależności i produkty pomiędzy testowaniem a innymi osobami lub działami
- Określenie, które elementy testowe wchodzą w zakres a które są poza zakresem testów na każdym z opisanych poziomów
- Kryteria wejścia, kryteria kontynuacji (kryteria wstrzymania/wznowienia testów), oraz kryteria wyjścia dla każdego poziomu, a także relacje między poziomami
- Ryzyko w projekcie testowym

W mniejszych projektach lub działaniach, mających tylko jeden sformalizowany poziom testowania, główny plan testów oraz plan testów dla tego właśnie poziomu łączy się zwykle w jeden dokument. Przykładowo, jeśli testowanie systemowe jest jedynym sformalizowanym poziomem testów, nieformalne testy modułowe oraz integracyjne wykonywane są przez programistów, a nieformalne testy akceptacyjne – przez klientów w ramach procesu testów beta, wówczas plan testów systemowych może zawierać elementy wymienione powyżej.

Co więcej, testowanie jest zwykle zależne od innych czynności w projekcie. Jeśli te czynności nie są dostatecznie udokumentowane, zwłaszcza pod kątem ich związków oraz wpływu na testowanie, wówczas niektóre dotyczące ich zagadnienia mogą pojawić się w głównym planie testów, lub w planie dla konkretnego poziomu testów.

Przykładowo, jeśli proces zarządzania konfiguracją nie jest udokumentowany, plan testów powinien określać, w jaki sposób zespołowi testowemu będą dostarczane elementy testowe.

3.2.4 Jednopoziomowy plan testów

Jednopoziomowy plan testów opisuje te czynności, które są wykonywane na danym poziomie testowania, uzupełniając – tam, gdzie to niezbędne – główny plan testów dla danego poziomu. Plan określa harmonogram, czynności oraz kamienie milowe, które nie są określone w głównym planie testów. Dodatkowo, o ile określone standardy oraz wzorce dokumentacji odnoszą się do testów na różnych poziomach, w jednopoziomowym planie testów wskazane zostaną standardy i wzorce używane w ramach konkretnego poziomu testów.

W mniej formalnych projektach i działaniach, plan testów jednego poziomu jest często jedynym dokumentem dotyczącym zarządzania testowaniem. W takich sytuacjach, niektóre z informacji wymienionych w podrozdziałach 3.2.1, 3.2.2 oraz 3.2.3, mogą wejść w skład tego dokumentu.

3.3 Szablony dokumentacji planu testów

Jak wspomniano w podrozdziale 3.2, szczegółowa struktura i zawartość głównego planu testów zależy od organizacji, od przyjętych przez nią standardów dokumentacji, oraz od stopnia sformalizowania projektu. Wiele organizacji wytwarza lub dostosowuje szablony (wśród nich również szablony planu testów) mające zapewnić jednolitość oraz czytelność dokumentacji pomiędzy różnymi projektami i działaniami.

Norma IEEE 829 “Standard for Software Test Documentation” zawiera szablony dokumentacji testowej (w tym plany testów) oraz wskazówki, jak się nimi posługiwać. Odnosi się ona również do tematu przekazywania obiektów testowych (czyli wydawania elementów testów do testowania).

3.4 Szacowanie testów

Szacowanie pracochłonności to czynność z zakresu zarządzania, która polega na określeniu kosztów oraz terminów ukończenia czynności wykonywanych w danym projekcie lub przedsięwzięciu. Najlepsze estymaty:

- reprezentują zebraną wiedzę doświadczonych praktyków i mają poparcie zaangażowanych osób;
- dostarczają jednoznacznego, szczegółowego zestawienia kosztów, zasobów, zadań oraz zaangażowanych osób;
- dla każdej oszacowanej czynności określają jej najbardziej prawdopodobny koszt, pracochłonność oraz czas trwania.

Szacowanie w inżynierii oprogramowania i systemów od dawna uważane jest za dziedzinę trudną, zarówno w sensie technicznym, jak i „politycznym”, mimo że rekomendowane praktyki szacowania są

dobrze określone w obszarze zarządzania projektami. Szacowanie testów polega na zastosowaniu najlepszych praktyk wobec czynności testowych w danym projekcie, bądź przedsięwzięciu.

Szacowanie testów powinno obejmować wszystkie czynności należące do procesu testowania, to znaczy planowanie i kontrolę testów, analizę i projektowanie testów, implementację i wykonywanie testów, ocenę i raportowanie testów, oraz czynności związane z zamknięciem testów. Szacunek kosztów, pracochłonności oraz – szczególnie – czasu trwania testów są przedmiotem szczególnego zainteresowania kierownictwa, gdyż wykonywanie testów zwykle znajduje się na ścieżce krytycznej projektu. Jednak zwykle szacowanie czasochłonności wykonywania testów jest trudne i może być niewiarygodne, gdy jakość oprogramowania jest niska lub w ogóle nieznana. Częstą praktyką jest szacowanie liczby wymaganych przypadków testowych. Przyjęte podczas szacowania założenia powinny być zawsze udokumentowane jako część estymaty.

Szacowanie testów powinno uwzględniać wszystkie czynniki mogące mieć wpływ na koszt, pracochłonność oraz czas trwania czynności testowych. Należą do nich między innymi następujące czynniki:

- Wymagany poziom jakości systemu
- Wielkość testowanego systemu
- Historyczne dane z wcześniejszych projektów testowych (także dane porównawcze)
- Czynniki procesowe, w tym: budowa strategii testów lub cykl życia w utrzymaniu oprogramowania i dojrzałość procesu; dokładność szacunków projektowych
- Czynniki materiałowe, takie jak: automatyzacja i narzędzia wspierające testowanie, środowiska wytwórcze, dane testowe, środowiska programistyczne, dokumentacja projektowa (np. wymagania, projekty itd.), oraz reużywalne artefakty testowe
- Czynniki ludzkie, m.in.: kierownicy oraz liderzy techniczni; zaangażowanie, wsparcie i oczekiwania wyższego kierownictwa; umiejętności, doświadczenie oraz postawy w zespole projektowym; niezmienność zespołu projektowego; relacje w zespole projektowym; wsparcie środowisk służących do testowania i debagowania; dostępność kompetentnych kontraktorów oraz konsultantów; wiedza dziedzinowa

Inne czynniki, mogące mieć wpływ na estymaty testów obejmują złożoność procesu, technologię, organizację, liczbę interesariuszy w obszarze testowania, liczbę zespołów (zwłaszcza rozproszonych geograficznie); znaczące potrzeby dotyczące wzrostu, szkoleń i orientacji; wdrożenie lub rozwój nowych narzędzi, technik, sprzętu dostosowywanego do potrzeb zamawiającego; liczbę testaliów; wymóg bardzo szczegółowej dokumentacji testowej, zwłaszcza zgodnej z nieznanym standardem dokumentacji; złożoność czasową dostaw komponentów, zwłaszcza w testach integracyjnych oraz podczas wytwarzania testów; ”wrażliwe” dane testowe (np. dane zależne od czasu).

Szacowania dokonuje się metodą wstępującą lub zstępującą. Do szacowania testowania stosuje się następujące techniki:

- Intuicja i zgadywanie
- Doświadczenia z przeszłości

- Struktura podziału prac (ang. *Work-Breakdown-Structure - WBS*)
- Sesje szacowania grupowego (np. ang. *wide band delphi*)
- Szacowanie trzypunktowe
- Analiza punktów testowych (TPA) [Pol02]
- Standardy i normy w danej organizacji
- Odsetek pracochłonności całego projektu lub w podziale na poszczególne profile (np. stosunek liczby testerów do liczby programistów)
- Historia i metryki w danej organizacji, w tym stworzone na podstawie metryk modele do szacowania liczby defektów, liczby cykli testowania, liczby przypadków testowych, średniej pracochłonności wykonania każdego testu, oraz liczby cykli testów regresywnych
- Średnie branżowe oraz modele pozwalające na przewidywanie, takie jak punkty testowe, punkty funkcyjne, linie kodu, szacowany nakład pracy programistów, oraz inne parametry projektowe.

Zwykle takie oszacowanie, po przygotowaniu, dostarczane jest wraz z uzasadnieniem kierownictwu (patrz podrozdział 3.7). Często następnym krokiem po estymacji są negocjacje, w wyniku których oszacowanie jest modyfikowane. Idealnym rozwiązaniem jest sytuacja, gdzie w ostatecznym oszacowaniu osiąga się równowagę pomiędzy celami organizacyjnymi i projektowymi w zakresie jakości, harmonogramu, budżetu oraz funkcjonalności.

3.5 Harmonogramowanie planowania testów

Planowanie umożliwia identyfikację i zarządzanie ryzykami powiązanymi z planowanymi czynnościami, staranne i wykonane na czas powiązanie planowanych czynności z innymi czynnościami oraz przygotowanie wysokiej jakości planu. Ta zasada dotyczy także planowania testowania. Jednak w przypadku planowania testowania, korzystając z oszacowań testowania, uzyskuje się dodatkowe korzyści, takie jak:

- Wykrycie i zarządzanie ryzykami projektowymi i innymi problemami, które występują poza obszarem samego testowania.
- Wykrycie i zarządzanie ryzykami i problemami produktowymi (jakościowymi) przed wykonywaniem testów.
- Identyfikacja problemów w planie projektu lub w innych artefaktach projektowych.
- Możliwość zwiększenia zespołu testowego, budżetu, pracochłonności lub czasu trwania testów w celu osiągnięcia wyższej jakości.
- Zidentyfikowanie komponentów krytycznych, co umożliwia przyśpieszenie ich dostaw.

Harmonogramowanie testów powinno być realizowane w powiązaniu z planowaniem wytwarzania, bowiem testowanie w dużym stopniu zależy od harmonogramu wytwarzania i dostaw.

Pożądane jest więc, aby wstępna wersja planów testów powstawała jak najwcześniej – w przeciwnym razie, zanim uzyska się pełną informację niezbędną do ukończenia planu testów, może być za późno, aby spożytkować związane z nim korzyści.

Kiedy docierają dalsze informacje, autor planu testów – zwykle kierownik testów – może je uwzględnić w planie testów. Taki iteracyjny sposób tworzenia, udostępniania i przeglądu planu testów jest także dobrą metodą na osiągnięcie porozumienia, komunikowanie się i prowadzenie dyskusji na temat testowania.

3.6 Monitorowanie i kontrola postępu testów

Postęp testów monitorowany jest w pięciu głównych wymiarach:

- Ryzyka produktowe
- Defekty
- Testy
- Pokrycie
- Zaufanie

W trakcie przedsięwzięcia lub projektu ryzyka produktowe, incydenty, testy oraz pokrycie testowe mogą być - i często są - mierzone i raportowane w określony sposób. Jest pożądane, aby te miary odwoływały się do kryteriów zakończenia określonych w planie testów. Zaufanie, choć można je mierzyć przy pomocy ankiet, zwykle raportuje się w sposób subiektywny.

Metryki odnoszące się do ryzyk produktowych obejmują:

- Liczbę otwartych ryzyk (włączając w to ich rodzaj i poziom)
- Liczbę zaadresowanych ryzyk (włączając w to ich rodzaj i poziom)

Metryki odnoszące się do defektów obejmują:

- Łączną liczbę zgłoszonych (zidentyfikowanych) defektów w porównaniu do łącznej liczby rozwiązanych (zamkniętych) defektów
- Średni czas pomiędzy awariami (MTBF) lub częstotliwość pojawiania się awarii (ang. *failure arrival rate*)
- Podział liczby defektów pod względem: określonych modułów lub elementów testów; przyczyn powstania, źródeł; wersji testowych; fazy powstania, wykrycia lub usunięcia; oraz - w pewnych przypadkach – właściciela
- Trend czasu liczonego od zgłoszenia do rozwiązania defektu

Metryki związane z testami obejmują:

- Łączną liczbę zaplanowanych, zaprojektowanych (skonstruowanych), wykonanych, zaliczonych, niezaliczonych, zablokowanych i pominiętych testów
- Status testów regresywnych i potwierdzających
- Liczba godzin testowania dziennie – porównanie czasu zaplanowanego z czasem rzeczywistym przeznaczonym na testy

Metryki odnoszące się do pokrycia testowego obejmują:

- Pokrycie wymagań i zaprojektowanych elementów
- Pokrycie ryzyka
- Pokrycie środowiska i konfiguracji

Te miary mogą być przekazywane w formie ustnej, pisemnej - liczbowo w tabelach lub w formie grafów i mogą być użyte do wielu celów, w tym:

- Analizy, by poprzez wyniki testów wykryć, co dzieje się z produktem, projektem lub procesem.
- Raportowania, aby przekazać to, co ujawniły testy, zainteresowanym członkom zespołu projektowego lub interesariuszom.
- Kontroli i nadzoru, aby zmienić przebieg testowania lub całego projektu i aby monitorować skutki takich zmian.

Właściwe sposoby gromadzenia, analizowania i raportowania tych miar testowych zależą od potrzeb informacyjnych, celów, oraz umiejętności ludzi posługujących się tymi miarami.

Stosując wyniki testów do pomiaru lub wpływania na czynności kontrolne w projekcie, należy uwzględnić następujące opcje:

- Rewizja analiz ryzyka jakościowego, priorytetów testowych i/lub planów testów
- Dodanie zasobów lub w inny sposób zwiększenie nakładów na testy
- Opóźnienie daty dostawy
- Obniżenie lub podwyższenie kryteriów zakończenia testowania

Wdrożenie takich opcji wymaga zwykle porozumienia między interesariuszami projektu lub przedsięwzięcia, oraz zgody kierownictwa projektu lub przedsięwzięcia.

Sposób zredagowania raportu testowego zależy przede wszystkim od osób, dla których jest on przeznaczony. Dla kierownika projektu zapewne przydatna jest szczegółowa informacja o defektach; dla kierownika obszaru biznesowego najważniejszym tematem raportowania może być status ryzyk produktowych.

Standard IEEE 829 „Standard for Software Test Documentation” zawiera szablon, w którym określono, jak raportować podsumowanie testów.

Kontrola testów powinna zostać wdrożona w oparciu o rozbieżności pomiędzy planem testów a raportem postępu testów. Celem kontroli testów jest zminimalizowane tych rozbieżności. Możliwe czynności kontrolne obejmują:

- Zmianę priorytetów przypadków testowych
- Pozyskanie dodatkowych zasobów
- Opóźnienie daty wydania

- Zmianę zakresu (funkcjonalności) projektu
- Zmianę kryteriów zakończenia testowania (wyłącznie za zgodą interesariuszy).

3.7 Biznesowa wartość testowania

Choć większość organizacji uważa testowanie za wartościowe, niewielu kierowników, w tym kierowników testów, potrafi określić liczbowo, opisać lub w inny sposób wyrazić tę wartość. Co więcej, wielu kierowników, liderów testów i testerów koncentruje się na taktycznych szczegółach testowania (specyficznych dla danej czynności lub poziomu), pomijając większe, strategiczne aspekty testowania (na wyższym poziomie), istotne dla pozostałych uczestników projektu, zwłaszcza dla kierownictwa.

Testowanie dostarcza wartości organizacji, projektowi lub przedsięwzięciu zarówno pod względem ilościowym, jak i jakościowym:

- Wartości ilościowe obejmują znajdowanie defektów, którym można zapobiec lub które można naprawić przed wydaniem; ograniczenie ryzyka dzięki wykonywaniu testów, oraz dostarczanie danych na temat statusu projektu, produktu i procesu.
- Wartości jakościowe obejmują poprawę opinii o poziomie jakości, sprawniejsze i bardziej przewidywalne wydania, wzrost zaufania, zabezpieczenie przed negatywnymi konsekwencjami prawnymi oraz ograniczenie ryzyka niepowodzenia całych przedsięwzięć lub nawet utraty ludzkiego życia.

Kierownicy i liderzy testów powinni rozumieć, które z tych wartości stosują się do ich organizacji, projektu lub przedsięwzięcia, i być w stanie przekazywać informację dotyczącą testowania w kontekście tych wartości.

Znaną metodą pomiaru wartości liczbowych i sprawności testowania jest tak zwany koszt jakości (lub niekiedy, koszt złej jakości). Koszt jakości polega na podziale kosztów projektu lub kosztów operacyjnych na cztery kategorie:

- Koszt zapobiegania
- Koszt wykrywania
- Koszty awarii wewnętrznej
- Koszty awarii zewnętrznej

Część budżetu testowego to koszty wykrywania, podczas gdy pozostała część to koszty wewnętrznej awarii. Łączne koszty wykrywania i wewnętrznej awarii zwykle leżą znacznie poniżej kosztów zewnętrznej awarii, co nadaje testowaniu znakomitą wartość. Określając koszty w tych czterech kategoriach, kierownicy i liderzy testów mogą stworzyć przekonywujące uzasadnienie biznesowe dla testowania.

3.8 Testowanie rozproszone, wykonywane przez osoby zakontraktowane oraz przez zewnętrzną firmę

Testowanie wykonywane jest często nie tylko przez pojedynczy zespół testowy, składający się z osób zatrudnionych na tych samych zasadach, zlokalizowany w jednym i tym samym miejscu, co reszta zespołu projektowego. Kiedy testowanie wykonywane jest w wielu miejscach, można je nazwać testowaniem rozproszonym. Testowanie wykonywane w więcej niż jednym miejscu, przez osoby, które nie są zatrudnione na tych samych zasadach, co reszta zespołu projektowego, pracujące w innym miejscu, możemy nazwać testowaniem wykonywanym przez firmę zewnętrzną (ang. *outsourcing*). Jeśli testowanie wykonują osoby zatrudnione na innych zasadach w stosunku do reszty zespołu projektowego, ale w tej samej lokalizacji, możemy je nazwać testowaniem wykonywanym przez osoby zakontraktowane (ang. *insourcing*).

Dla wszystkich tego rodzaju sposobów organizacji testowania istnieje wspólna potrzeba jasno określonych metod porozumiewania się i dobrze zdefiniowanych oczekiwań wobec przedsięwzięć, zadań i produktów. Zespół projektowy musi mniej polegać na nieformalnych kanałach komunikacji, takich jak rozmowy na korytarzach i czas spędzany wspólnie z kolegami z projektu. Położenie, strefa czasowa, różnice językowe i kulturowe powodują, że zagadnienia te stają się jeszcze ważniejsze.

Wspólne dla tego typu przedsięwzięć jest także uzgodnienie metodyk. Jeśli dwa zespoły testowe posługują się różnymi metodykami albo zespół testowy używa innej metodyki niż programiści lub kierownictwo projektu, może to spowodować znaczne problemy, zwłaszcza podczas wykonywania testów.

W testowaniu rozproszonym, podział zadań między różnymi lokalizacjami musi być jednoznaczny i przemyślany. Bez takich wskazówek, nawet najlepsza grupa testowa może nie wykonać pracy, do której jest znakomicie wykwalifikowana. Co więcej, testowanie jako całość będzie miało luki (co oznacza wzrost ryzyka dostawy o obniżonej jakości) lub zadania będą się niepotrzebnie nakładać (zmniejszenie efektywności).

Na koniec, w tego typu przedsięwzięciach testowych, najważniejsze jest, aby cały zespół projektowy zdobył i zachował zaufanie co do tego, że wszystkie zespoły testowe wykonają swoje zadania poprawnie, mimo ograniczeń organizacyjnych, kulturowych, językowych i geograficznych. Brak zaufania prowadzi do nieefektywności i opóźnień związanych z weryfikacją zadań, wzajemnego obwiniania się i politycznych gier w organizacji.

3.9 Testowanie na podstawie ryzyka

3.9.1 Wstęp do testowania na podstawie ryzyka

Ryzyko to prawdopodobieństwo wystąpienia niepożądanego wyniku. Ryzyka istnieją tam, gdzie może pojawić się jakiś problem, który obniży ocenę jakości produktu przez klienta, użytkownika, lub innego interesariusza, bądź zmniejszy szansę powodzenia projektu.

W sytuacji, gdy potencjalny problem dotyczy przede wszystkim jakości produktu, ryzyko nazywane jest produktowym (albo ryzykiem jakościowym). Przykładem może być defekt w niezawodności (błąd) powodujący awarię systemu podczas jego normalnego działania. W przypadku, gdy potencjalny problem dotyczy przede wszystkim powodzenia projektu, ryzyko nazywane jest projektowym (albo ryzykiem planowania). Przykładem może być brak personelu, co może spowodować opóźnienie projektu.

Nie wszystkie zagrożenia są równie istotne. Poziom ryzyka zależy od różnych czynników:

- Prawdopodobieństwa zaistnienia problemu
- Wpływu problemu, jeśli zaistnieje

W testowaniu na podstawie ryzyka, testowanie jest wykonywane w sposób, który adresuje ryzyko:

- Nakład pracy na poszczególne działania, wybór technik, kolejność działań testowych i usuwanie defektów wykonuje się w sposób dostosowany do poziomu ryzyka dotyczącego każdego ważnego, zidentyfikowanego ryzyka produktowego (jakościowego).
- Planowanie i zarządzanie pracą testową uwzględnia potrzeby zapobiegania oraz działań zaradczych dla każdego ważnego, zidentyfikowanego ryzyka projektowego (planowania).
- Wyniki testowe oraz status projektu podaje się tak, by ujawniały poziom niezaadresowanego ryzyka, na przykład wykorzystując informacje o testach, które nie zostały jeszcze wykonane lub testach pominiętych, albo informacje o defektach, które jeszcze nie zostały usunięte lub zretestowane.

Te trzy formy odpowiedzi na ryzyko powinny pojawiać się w trakcie całego cyklu życia projektu, nie tylko na początku i na końcu projektu. W szczególności, podczas trwania projektu, testerzy powinni dokładać starań, aby:

- Ograniczyć ryzyko znajdując najważniejsze defekty (jeśli chodzi o ryzyko produktowe) i wdrażając stosowne czynności zapobiegawcze lub korygujące, określone wcześniej w strategii testów i w planie testów.
- Oszacować zidentyfikowane ryzyko, korygować jego prawdopodobieństwo lub skutki w oparciu o nowe informacje napływające z projektu.

W obu przypadkach podjęte działania wpływają na to, jak testowanie reaguje na ryzyko.

Testowanie na podstawie ryzyka pod wieloma względami podobne jest do ubezpieczenia. Kupuje się ubezpieczenie na wypadek zaistnienia ryzyk, których się obawiamy, a jednocześnie ignorujemy te, których się nie boimy.

Analiza ilościowa, podobna do szacowania ryzyka stosowanego przez profesjonalistów kalkulujących ryzyko z branży reasekuracji oraz przez innych specjalistów ubezpieczeniowych, może znaleźć zastosowanie w testowaniu na podstawie ryzyka, ale zwykle testowanie na podstawie ryzyka bazuje na analizie jakościowej.

Aby móc poprawnie przeprowadzić testowanie na podstawie ryzyka, testerzy powinni umieć rozpoznawać, analizować i zapobiegać typowym zagrożeniom produktowym i projektowym, związanym z bezpieczeństwem, z czynnikami biznesowymi i ekonomicznymi, zabezpieczeniami systemu i danych, a także czynnikami organizacyjnymi i technicznymi.

3.9.2 Zarządzanie ryzykiem

Zarządzanie ryzykiem można sobie wyobrazić jako zadanie składające się z trzech podstawowych czynności:

1. Identyfikacji ryzyka
2. Analizy ryzyka
3. „Łagodzenia" ryzyka (zwanego też kontrolą ryzyka)

W pewnym stopniu czynności te wykonuje się sekwencyjnie, ale konieczność ciągłego zarządzania ryzykiem, wspomniana w poprzednim i w dalszych akapitach, oznacza że podczas projektu wszystkie czynności zarządzania ryzykiem wykonuje się iteracyjnie.

Najskuteczniejsze zarządzanie ryzykiem wymaga udziału wszystkich interesariuszy, choć realia projektu wymuszają niekiedy, że jedni interesariusze reprezentują innych. Przykładowo, w produkcji oprogramowania na rynki masowe, niekiedy prosi się niewielką grupę potencjalnych klientów o znalezienie możliwych defektów mających największy wpływ na używanie przez nich danego oprogramowania. W takim przypadku próbka potencjalnych klientów służy jako reprezentacja wszystkich możliwych klientów.

Ze względu na swoją wiedzę, analitycy testów powinni brać aktywny udział w procesie identyfikacji oraz analizy ryzyka.

3.9.2.1 Identyfikacja ryzyka

Identyfikując ryzyka produktowe i projektowe, testerzy mogą skorzystać z jednej z następujących technik:

- Wywiady eksperckie
- Niezależne oceny
- Użycie wzorców ryzyk
- Wyciąganie wniosków np. w sesji oceny projektu
- Warsztaty ryzyka (np. FMEA)
- Burza mózgów
- Listy kontrolne
- Wykorzystanie wcześniejszych doświadczeń

Proces rozpoznawania ryzyka wykryje najwięcej znaczących zagrożeń, jeśli odwoła się do jak najszerszej grupy interesariuszy.

Niektóre podejścia do rozpoznawania ryzyka kończą cały proces na samym rozpoznaniu ryzyka.

Niektóre techniki, takie jak analiza przyczyn i skutków awarii FMEA wymagają, aby dla każdej potencjalnej przyczyny awarii określić jej wpływ na resztę systemu (także na poziom najwyższy w odniesieniu do systemu systemów) oraz wpływ na potencjalnych użytkowników systemu.

Inne techniki, takie jak analiza zagrożeń (ang. *Hazard Analysis*), wymagają określenia źródła ryzyka.

Opis zastosowania metod analiza przyczyn i skutków awarii (FMEA) i analiza przyczyn, skutków i krytyczności awarii (FMECA) przedstawiony jest w podrozdziale 3.10 oraz w [Stamatis95], [Black02], [Craig02], [Gerrard02].

3.9.2.2 Analiza ryzyka

Podczas, gdy identyfikacja ryzyka polega na rozpoznawaniu jak największej liczby możliwych zagrożeń, analiza ryzyka polega na ich zbadaniu. W szczególności, na zaklasyfikowaniu każdego ryzyka oraz określeniu jego prawdopodobieństwa i skutków.

Zaklasyfikowanie ryzyka polega na przypisaniu go do właściwej kategorii. Typowe kategorie ryzyk omawiane są w normie ISO 9126, dotyczącej właściwości jakości. Niektóre organizacje wyodrębniają własne zestawy cech jakościowych. Warto zauważyć, że dokonując rozpoznania ryzyka przy pomocy listy kontrolnej, przypisanie go do kategorii następuje równocześnie z samą identyfikacją.

Określenie poziomu (lub wagi) ryzyka oznacza zwykle ocenę prawdopodobieństwa wystąpienia tego ryzyka oraz jego ewentualnych skutków. Prawdopodobieństwo wystąpienia często interpretuje się jako prawdopodobieństwo zaistnienia potencjalnego problemu w testowanym systemie. Innymi słowy, jego źródłem jest ryzyko techniczne.

Czynniki wpływające na ryzyko techniczne to:

- Złożoność technologii oraz zespołów
- Kwestie personalne oraz dotyczące szkoleń pomiędzy analitykami biznesowymi, projektantami i programistami
- Konflikty w zespole
- Problemy z dostawcami
- Rozproszenie geograficzne organizacji wytwarzającej oprogramowanie
- Podejścia zastane a nowe
- Narzędzia i technologia
- Złe zarządzanie lub kierownictwo techniczne
- Brak środków, brak czasu oraz naciski ze strony kierownictwa
- Brak zapewnienia jakości we wcześniejszych etapach
- Częste zmiany

- Wysoki współczynnik defektów we wcześniejszych etapach
- Zagadnienia związane z interfejsami oraz integracją

Skutek wystąpienia zagrożenia zwykle określa się jako jego wagę dla użytkowników, klientów lub innych interesariuszy. Innymi słowy, źródłem skutków jest ryzyko biznesowe. Czynniki wpływające na ryzyko biznesowe to:

- Częstotliwość używania funkcjonalności, której dotyczy ryzyko
- Szkody dla wizerunku
- Utrata biznesu
- Potencjalne finansowe, ekologiczne lub społeczne straty, bądź odpowiedzialność prawna
- Sankcje cywilnoprawne lub karne
- Utrata licencji
- Brak rozsądnych możliwości obejścia
- Widoczność awarii prowadząca do negatywnego rozgłosu

Z punktu widzenia testowania, poziom ryzyka można określać ilościowo lub jakościowo. Jeżeli prawdopodobieństwo oraz wpływ daje się oszacować ilościowo, można te dwie wartości pomnożyć, aby wyliczyć koszt materializacji ryzyka. Ten koszt oznacza prawdopodobną stratę, związaną z danym ryzykiem.

Zwykle jednak poziom ryzyka daje się określić tylko jakościowo. Oznacza to, że można mówić o prawdopodobieństwie bardzo wysokim, wysokim, średnim, niskim lub bardzo niskim, ale nie da się stwierdzić z pewnością, czy wynosi ono 90%, 75%, 50%, 25%, czy 10%. Podejścia jakościowego nie należy traktować jako gorszego od podejścia ilościowego. W rzeczywistości, źle zastosowane podejście ilościowe wprowadza interesariuszy w błąd co do tego, na ile rzeczywiście rozumiemy i zarządzamy ryzykiem. Nieformalne metody, takie jak te opisane w [vanVeenendaal02], [Craig02] i [Black07b] są zwykle jakościowe i mniej rygorystyczne.

Jeśli analiza ryzyka nie jest wykonywana na podstawie obszernych, statystycznie istotnych danych (tak jak postępuje się w firmach ubezpieczeniowych), niezależnie od tego, czy jest to analiza jakościowa czy ilościowa, będzie ona bazować na subiektywnej ocenie prawdopodobieństwa i wpływu. Oznacza to, że na określany poziom ryzyka wpływa indywidualny odbiór i ocena tych czynników. Kierownicy projektów, programiści, użytkownicy, analitycy biznesowi, architekci i testerzy mają zwykle odmienne punkty widzenia i wobec tego niekiedy różne opinie na temat poziomu ryzyka dla każdego elementu ryzyka. Proces analizy ryzyka powinien więc zawierać metodę osiągania porozumienia lub, w najgorszym scenariuszu, odgórnego ustalenia poziomu ryzyka. W przeciwnym razie, nie da się używać informacji o poziomie ryzyka jako wytycznych dla czynności łagodzenia ryzyka.

3.9.2.3 Łagodzenie ryzyka

Po zidentyfikowaniu i przeanalizowaniu ryzyka, istnieją cztery podstawowe sposoby radzenia sobie z ryzykiem:

1. Ograniczenie ryzyka przy pomocy czynności zapobiegawczych, zmniejszających prawdopodobieństwo lub skutków jego wystąpienia.
2. Sporządzenie planów awaryjnych, ograniczających wpływ ryzyka, kiedy staje się ono rzeczywistością.
3. Przekazanie ryzyka do rozwiązania innej stronie.
4. Zignorowanie i zaakceptowanie ryzyka.

Wybór jednej z powyższych opcji zależy od korzyści i możliwości oraz kosztów każdej z nich, jak również możliwych dodatkowych zagrożeń z nimi związanych.

Strategie zapobiegania

W większości strategii testowych na podstawie ryzyka, identyfikacja ryzyka, analiza ryzyka i ustalenie czynności minimalizujących jego wpływ są podstawą głównego planu testów i innych planów testów. Poziom ryzyka związanego z każdym elementem ryzyka określa pracochłonność testów (to znaczy czynności zapobiegawczych) związanych z każdym ryzykiem. Niektóre standardy dotyczące bezpieczeństwa (np. FAA DO-178B/ED 12B, IEC 61508) określają techniki testowania oraz poziom pokrycia na podstawie poziomu ryzyka.

Łagodzenie ryzyka projektowego

Zidentyfikowane ryzyka powinny być komunikowane kierownikowi projektu, który być może będzie musiał podjąć działania w oparciu o otrzymane informacje. Organizacja wykonująca testy nie zawsze jest w stanie zminimalizować każde z takich zagrożeń. Jednak niektóre z nich kierownik testów może skutecznie ograniczać, na przykład:

- Gotowość środowiska testowego i narzędzi
- Kwalifikacje i dostępność personelu testowego
- Niska jakość produktów wejściowych do testowania
- Zbyt duża zmienność produktów dostarczanych do testowania
- Brak standardów, reguł i technik testowania

W skład metod łagodzenia ryzyka wchodzi wczesne przygotowywanie testaliów, testy wstępne sprzętu testowego, testy wstępne wykonywane na wcześniejszych wersjach produktu, ostrzejsze kryteria wejściowe do testowania, wymagania na testowalność, udział w przeglądach wcześniejszych produktów projektu, udział w zarządzaniu problemami oraz zmianami, oraz monitorowanie postępów i jakości projektu.

Zapobieganie ryzyku produktowemu

Kiedy mówi się o ryzyku produktowym (jakościowym), wtedy testowanie jest metodą na zapobieganie takim zagrożeniom. Znajdując defekty, testerzy ograniczają ryzyko, uświadamiając istnienie błędów oraz stwarzając możliwości radzenia sobie z nimi przed wydaniem. W sytuacji, gdy defekty nie zostają znalezione, testowanie ogranicza ryzyko dając zapewnienie, że w pewnych – przetestowanych – warunkach, system funkcjonuje poprawnie.

Ryzyku produktowemu (jakościowemu) można też zapobiegać czynnościami innymi, niż testowanie. Przykładowo, jeśli zidentyfikowano, że wymagania nie są opisane w dobry sposób, efektywnym rozwiązaniem byłoby wprowadzenie dokładnych przeglądów (w przeciwieństwie do tworzenia i priorytetyzowania testów, które zostałyby wykonane już po stworzeniu złego projektu i złego kodu na podstawie źle napisanych wymagań).

Informacji o ryzyku używa się czasami do nadawania testom priorytetów. Testy najwyższego ryzyka wykonywane są przed testami niskiego ryzyka. Testy wykonuje się w ściśle określonej kolejności (często jest to nazywane podejściem „głębokim"). Niekiedy stosuje się podejście próbkowania, polegające na wybraniu próby testów dla wszystkich zidentyfikowanych ryzyk, stosując poziom ryzyka do kształtowania udziału poszczególnych zagrożeń w próbce, a jednocześnie gwarantując, że każde ryzyko zostanie choć raz pokryte testami (często nazywane podejściem „szerokim").

Wśród innych sposobów łagodzenia skutków zagrożeń znajdują się:

- Wybór właściwej techniki projektowania testów
- Przeglądy oraz inspekcje
- Przeglądy projektu testów
- Poziom niezależności
- Wykorzystanie najbardziej doświadczonych osób
- Sposób wykonywania retestów
- Testowanie regresywne

W [Gerrard02] wprowadza się pojęcie skuteczności testowania, jako miary (procentowej) wskazującej, jak skuteczne jest testowanie jako środek ograniczania ryzyka. Nie stosuje się testowania do ograniczania ryzyka, gdy jego skuteczność w tym zakresie jest mała. Niezależnie od tego, czy testy na podstawie ryzyka stosuje się w sposób „głęboki", czy w sposób „szeroki", może się zdarzyć, że czas przeznaczony na testy nie starczy na ich wykonanie. W takiej sytuacji testowanie na podstawie ryzyka umożliwia testerom raportowanie kierownictwu poziomu pozostałego w danym momencie ryzyka, co pozwala na podjęcie decyzji, czy należy przedłużyć testowanie, czy też przenieść pozostałe ryzyko na użytkowników, klientów, pracowników wsparcia powdrożeniowego/technicznego lub operacyjnych.

Dopasowanie testowania do kolejnych cykli testowania

Jeśli dostępny jest czas na dalsze testowanie, kolejne cykle testowe trzeba dostosować do wyników nowej analizy ryzyka. Dotyczy to głównie zupełnie nowych lub znacznie zmienionych zagrożeń

produktowych, takich jak: niestabilne lub pełne defektów obszary zidentyfikowane podczas testowania; zagrożenia biorące się z naprawionych defektów; wysiłki, aby skupić testowanie na typowych usterkach znajdowanych podczas testowania; oraz - być może - niedostatecznie przetestowane obszary (małe pokrycie testowe). Na podstawie analizy takich zagrożeń należy planować każdy kolejny lub dodatkowy cykl testów. Zaleca się również, aby nową wersję arkusza zagrożeń tworzyć przy każdym kamieniu milowym projektu.

3.9.3 Zarządzanie ryzykiem w cyklu życia

W sytuacji idealnej, zarządzanie ryzykiem ma miejsce podczas całego cyklu życia. Jeśli w organizacji istnieje udokumentowana polityka i/lub strategia testów, powinny one opisywać podejście do ryzyka produktowego i projektowego w testowaniu, sposób, w jaki zarządzanie ryzykiem włączone jest we wszystkie etapy testowania oraz jak na nie wpływa.

Czynności związane z identyfikacją i analizą ryzyka mogą rozpocząć się już w początkowej fazie projektu, niezależnie od tego, według jakiego modelu rozwoju oprogramowania projekt jest realizowany. Czynności mające na celu zapobieganie ryzyku można zaplanować i wdrożyć w ramach ogólnego procesu planowania testów. Główny plan testów lub plany testów poszczególnych poziomów mogą odnosić się do zagrożeń, tak produktowych, jak i projektowych. Rodzaj i poziom ryzyka wpływają na te poziomy testów, gdzie podejmuje się związane z danym ryzykiem działania, na zakres przedsięwzięć mających na celu zapobieganie ryzyku, na techniki testowe i inne metody stosowane w celu ograniczenia ryzyka, oraz na kryteria, przy pomocy których ocenia się skuteczność podjętych środków zapobiegawczych.

Po zakończeniu planowania, w projekcie powinno cały czas mieć miejsce zarządzanie ryzykiem, obejmujące jego identyfikację, analizę i łagodzenie skutków. Zawiera się w tym identyfikacja nowych ryzyk, ponowna ocena poziomu istniejących zagrożeń, oraz szacowanie skuteczności czynności zapobiegających zagrożeniom. Przykładowo, jeśli w fazie pracy z wymaganiami odbyła się sesja identyfikowania i analizy zagrożeń na podstawie specyfikacji wymagań, to powinno się taką sesję powtórzyć po sfinalizowaniu projektu rozwiązania.

Inny przykład: jeśli podczas testowania modułu zostaje wykryte więcej defektów niż oczekiwano, oznacza to, że w tym obszarze prawdopodobieństwo defektów jest wyższe, niż przewidywane, a więc należy zwiększyć także prawdopodobieństwo i poziom ryzyka. W wyniku tego może się zwiększyć ilość testów, które należy wykonać wobec tego modułu.

Po zakończeniu wstępnej identyfikacji i analizy ryzyka oraz po przeprowadzeniu czynności zmniejszających ryzyko, można zmierzyć, na ile ryzyko zostało obniżone. Można to osiągnąć, wiążąc przypadki testowe i znajdowane defekty z zagrożeniami, do których się odnoszą. Wykonując testy i znajdując defekty, testerzy mogą analizować poziom pozostałego ryzyka, co ułatwia wykorzystanie testowania na podstawie ryzyka do określenia właściwego terminu wydania (wersji). Przykład, jak raportować wyniki testów w odniesieniu do pokrycia ryzyka, jest opisany w [Black03]. Raportowanie testów powinno odnosić się zarówno do zagrożeń już pokrytych testami, jak i do tych nadal otwartych, a także do osiągniętych i jeszcze nieosiągniętych korzyści.

3.10 Analiza przyczyn i skutków awarii

Analiza przyczyn i skutków awarii FMEA oraz jej wersja uwzględniająca analizę krytyczności FMECA - akronim od: *Failure Mode, Effect and Criticality Analysis*, to powtarzalne czynności, których celem jest analiza skutków oraz krytyczności sytuacji awaryjnych w systemie. Zastosowanie tych metod analizy wobec oprogramowania jest niekiedy określane jako SFMEA i odpowiednio SFMECA, gdzie litera "S" na początku oznacza oprogramowanie (ang. *software*). Kolejne podrozdziały opisują wyłącznie FMEA, ale stosują się także do pozostałych trzech metod.

Testerzy muszą umieć współpracować przy tworzeniu dokumentu analizy przyczyn i skutków awarii, co wymaga zarówno zrozumienia celu i zastosowania tego dokumentu, jak i umiejętności zastosowania swojej wiedzy do określenia czynników ryzyka.

3.10.1 Obszary zastosowania

FMEA powinna być stosowana:

- Tam, gdzie poziom krytyczności budowanego oprogramowania lub systemu musi być analizowany w celu obniżenia ryzyka wystąpienia awarii (np. systemy krytyczne pod względem bezpieczeństwa, takie jak systemy kontroli lotu samolotów)
- Tam, gdzie wymagają tego regulacje prawne (por. podrozdział 1.3.2 dotyczący systemów krytycznych pod względem bezpieczeństwa)
- By usunąć defekty we wczesnych fazach
- By określić szczególne wymagania testowe, ograniczenia operacyjne lub decyzje dotyczące budowy systemu, dokonane wobec systemów krytycznych pod względem bezpieczeństwa.

3.10.2 Fazy implementacji

Analizę przyczyn i skutków awarii należy rozpocząć jak tylko dostępna jest ogólna informacja i rozszerzać ją, gdy tylko otrzyma się więcej szczegółów. FMEA można stosować na każdym poziomie dekompozycji systemu lub oprogramowania, w zależności od dostępnej informacji oraz wymagań wobec programu.

Wobec każdej krytycznej funkcji, modułu lub komponentu wielokrotnie wykonuje się:

- Wskazanie możliwych trybów awarii funkcji, czyli sposobów, w jaki może dojść do awarii
- Określenie możliwych przyczyn tych awarii, ich skutków, oraz zaprojektowanie mechanizmów zapobiegających, lub ograniczających skutki tych awarii.

3.10.3 Korzyści i koszty

Stosowanie metody analizy przyczyn i skutków awarii FMEA przynosi następujące korzyści:

- Umożliwia wykrycie awarii systemu spowodowanych awariami oprogramowania lub jego błędnym stosowaniem

- Daje wsparcie dla analizy systemowej, przy systematycznym stosowaniu
- Wyniki FMEA znajdują zastosowanie przy podejmowaniu decyzji projektowych oraz ich uzasadnianiu
- Wyniki FMEA można stosować, aby skupić testowanie wokół określonych (krytycznych) obszarów oprogramowania
- Stosując FMEA należy uwzględnić następujące czynniki:
  - Rzadko bierze się pod uwagę sekwencje awarii
  - Tworzenie tabel FMEA jest czasochłonne
  - Niezależne funkcje czasami trudno określić
  - Trudno jest zidentyfikować skutki propagacji błędów

3.11 Zastosowanie zarządzania testowaniem w różnych obszarach

3.11.1 Zarządzanie testami a testowanie eksploracyjne

Zarządzanie testowaniem w sesjach (ZTWS) odnosi się do zarządzaniem testami eksploracyjnymi. Sesją nazywana jest podstawowa, nieprzerwana jednostka pracy testowej, dotycząca jednego obiektu testowego i określonego celu testowania (tak zwany statut testu). Na zakończenie każdej sesji sporządzany jest raport, zwany arkuszem sesji. ZTWS funkcjonuje w ramach udokumentowanego procesu, a jej zgłoszone wyniki uzupełniają dokumentację weryfikacji.

Sesję testu dzieli się na trzy etapy:

- Przygotowanie sesji: przygotowanie środowiska testowego oraz lepsze zrozumienie produktu
- Projektowanie i wykonywanie: uważne badanie obiektu testów i poszukiwanie problemów
- Badanie defektów oraz raportowanie: rozpoczyna się, gdy tester zidentyfikuje przypuszczalną awarię.

Arkusz sesji ZTWS zawiera:

- Statuty sesji
- Nazwiska testerów
- Datę i czas rozpoczęcia sesji
- Podział zadań (sesje)
- Pliki danych
- Notatki
- Kwestie
- Defekty

Na zakończenie każdej sesji kierownik testów omawia ją z jej uczestnikami. Podczas tego spotkania, kierownik dokonuje przeglądu arkusza sesji, udoskonala statuty (ang. *charters*), otrzymuje informację od testerów oraz oszacowuje i planuje kolejne sesje.

Agenda opisywanego powyżej spotkania obejmuje następujące elementy (w skrócie PROOF od pierwszych liter ich angielskich nazw):

- Przeszłość (ang. *Past*): co zaszło podczas sesji?

- Rezultaty (ang. *Results*): co osiągnięto w czasie sesji?
- Plany (ang. *Outlook*): co jeszcze należy zrobić?
- Przeszkody (ang. *Obstacles*): co utrudniało dobre testowanie?
- Odczucia (ang. *Feelings*): co tester myśli o sesji?

3.11.2 Zarządzanie testowaniem a systemy systemów

Przy zarządzaniu testowaniem w odniesieniu do systemu systemów bierze się pod uwagę następujące zagadnienia:

- Zarządzanie jest bardziej złożone, bowiem testowanie poszczególnych systemów systemu może być przeprowadzane w różnych miejscach, przez różne organizacje, w oparciu o różne modele cyklu życia oprogramowania. Z tych względów główny plan testowy systemu zbudowanego z systemów zwykle wprowadza formalny cykl życia oprogramowania, podkreślający elementy służące przede wszystkim zarządzaniu, takie jak kamienie milowe i punkty kontroli jakości. Często stosuje się także formalnie zdefiniowany proces kontroli jakości, który może być opisany w odrębnym planie jakości.
- Procesy wspomagające, takie jak zarządzanie konfiguracją, zmianami i wydaniami, jak również ich powiązanie z zarządzaniem testowaniem, muszą być formalnie zdefiniowane. Procesy te są niezbędne, aby móc nadzorować dostawy i zmiany oprogramowania, tak, aby testowanie odbywało się na określonych wersjach oprogramowania.
- Budowa i zarządzanie reprezentatywnymi środowiskami testowymi z uwzględnieniem danych testowych bywają trudne, zarówno organizacyjnie, jak i technicznie.
- Strategia testów integracyjnych może wymagać stosowania symulatorów. O ile może to być stosunkowo proste i niedrogie dla testów integracyjnych na wcześniejszych poziomach testowych, o tyle budowa symulatorów całych systemów może być złożona i kosztowna na wyższych poziomach integracji, z którymi mamy do czynienia w systemach systemów. Planowanie, szacowanie i budowanie symulatorów często organizuje się jako odrębny projekt.
- Podczas testowania systemów, zależności między poszczególnymi częściami powodują dodatkowe ograniczenia w testach systemowych i akceptacyjnych. Wymaga to również poświęcenia dodatkowej uwagi integracyjnym testom systemowym oraz związanym z nimi dokumentami, stanowiącymi podstawę testów, na przykład specyfikacjom interfejsów.

3.11.3 Zarządzanie testowaniem a systemy krytyczne pod względem bezpieczeństwa

Z zarządzaniem testowaniem w odniesieniu do systemów krytycznych ze względu na bezpieczeństwo wiążą się następujące zagadnienia:

- Dla systemów tych istnieją zwykle standardy specyficzne dla danej gałęzi przemysłu (dziedzinowe) – przykładem są tutaj przemysły transportowy, medyczny i wojskowy. Mogą się one odnosić do procesu wytwarzania, do struktury organizacyjnej, albo do wytwarzanego produktu. Więcej szczegółów na ten temat znajduje się w rozdziale 6.
- Z powodu odpowiedzialności prawnej, związanej z systemami krytycznymi pod względem bezpieczeństwa, dotyczyć ich mogą wymagania, których spełnienie trzeba potwierdzić w celu zademonstrowania zgodności z regulacjami prawnymi. Wymagania te mogą dotyczyć

możliwości śledzenia powiązań wymagań, niezbędnego poziomu pokrycia testowego, kryteriów zakończenia testów i wymaganej dokumentacji testowej.
- Aby wykazać zgodność struktury organizacyjnej oraz procesu wytwarzania, wystarczające mogą być audyty oraz schematy organizacyjne.
- Proces wytwarzania wykonywany jest zgodnie z określonym z góry cyklem życia, zależnie od standardu, który go dotyczy. Takie cykle życiowe są z reguły sekwencyjne.
- Dla systemu, zakwalifikowanego przez organizację jako krytyczny, konieczne jest odniesienie się w strategii i/lub w planie testów do:
  - Niezawodności
  - Dostępności
  - Łatwości utrzymania
  - Bezpieczeństwa oraz zabezpieczeń

Z powodu tych atrybutów, takie systemy nazywane są czasem systemami RAMS (od angielskich słów *Reliability, Availability, Maintainability, Safety* i *Security*).

3.11.4 Inne zagadnienia dotyczące zarządzania testowaniem

Znacznym zagrożeniem dla powodzenia wytwarzania aplikacji jest niezaplanowanie testów niefunkcjonalnych. Jednak wiele rodzajów testów niefunkcjonalnych kojarzonych jest z wysokimi kosztami, które trzeba zrównoważyć z powiązanym z nimi ryzykiem.

Istnieje wiele rodzajów testów niefunkcjonalnych i nie wszystkie są odpowiednie dla danej aplikacji. Na planowanie i wykonanie testów niefunkcjonalnych wpływają następujące czynniki:

- Wymagania interesariuszy
- Wymagane narzędzia
- Wymagana platforma sprzętowa
- Czynniki organizacyjne
- Komunikacja
- Bezpieczeństwo danych

3.11.4.1 Wymagania interesariuszy

Wymagania niefunkcjonalne często są słabo określone lub pomijane. W fazie planowania, testerzy muszą pozyskać informacje o oczekiwaniach interesariuszy i ocenić związane z nimi ryzyka.

Podczas określania wymagań zalecane jest uzyskanie informacji z różnych źródeł. Wymagania muszą być pozyskiwane z takich źródeł jak klienci, użytkownicy, pracownicy operacyjni i pracownicy zespołów utrzymaniowych, w przeciwnym razie niektóre wymagania mogą zostać pominięte.

Aby ułatwić testowanie wymagań niefunkcjonalnych, trzeba uwzględnić następujące zasady:

- Wymagania powinny być częściej czytane niż pisane. Wysiłek włożony w utworzenie testowalnych wymagań jest niemal zawsze opłacalny. Należy posługiwać się prostym,

spójnym i zwięzłym językiem, na przykład językiem zdefiniowanym w słowniku projektu. W szczególności, należy z rozmysłem używać słów takich jak „ma” (w znaczeniu „musi”), „powinien” (w znaczeniu „jest pożądany”) oraz „musi” (najlepiej go unikać lub używać jako synonimu słowa „ma”).

- Osoby, które czytają wymagania, powinny wywodzić się z różnych dziedzin.
- Wymagania muszą być pisane jasno i zwięźle, aby uniknąć różnych interpretacji. Do opisu wszystkich wymagań należy używać standardowego formatu.
- Kiedy to tylko możliwe, wymagania powinno się określać liczbowo. Należy znaleźć stosowną miarę do wyrażenia wartości danego atrybutu (np. osiągi mierzone w milisekundach) i określić przedział wartości dla zaakceptowania lub odrzucenia wyników rzeczywistych. Nie jest to łatwym zadaniem wobec niektórych niefunkcjonalnych atrybutów, na przykład dla użyteczności.

3.11.4.2 Niezbędne wsparcie narzędziowe

Narzędzia komercyjne lub symulatory są szczególnie przydatne do testów wydajności, efektywności oraz niektórych testów bezpieczeństwa. Planowanie testów powinno obejmować oszacowanie kosztów i harmonogram wdrożenia narzędzi. Tam, gdzie mają być stosowane specjalistyczne narzędzia, podczas planowania należy uwzględnić również krzywą uczenia się stosowania narzędzi lub koszt zatrudnienia zewnętrznych specjalistów.

Budowa złożonego symulatora może stanowić osobny projekt wytwórczy i powinno się ją w ten sposób planować. W szczególności, planowanie symulatorów wykorzystywanych w testach aplikacji krytycznych pod względem bezpieczeństwa, powinno brać pod uwagę testy akceptacyjne oraz ewentualną certyfikację symulatora przez niezależną instytucję.

3.11.4.3 Wymagana platforma sprzętowa

Aby dostarczyć realistycznych wyników, wiele testów niefunkcjonalnych wymaga środowiska zbliżonego do docelowego środowiska produkcyjnego. Może to mieć – w zależności od rozmiarów i złożoności testowanego systemu - znaczny wpływ na planowanie i finansowanie testów. Koszt wykonania testów niefunkcjonalnych bywa tak wysoki, że pozostaje niewiele czasu na wykonanie testów.

Przykładowo, weryfikacja wymagania na skalowalność często odwiedzanego portalu internetowego może wymagać symulowania setek tysięcy użytkowników wirtualnych. Może to mieć znaczący wpływ na koszty narzędzi i sprzętu. Ponieważ koszty te zwykle minimalizuje się wypożyczając narzędzia, czas dostępny na tego typu testy jest ograniczony.

Wykonywanie testów użyteczności może wymagać budowy specjalnie przeznaczonych do tego laboratoriów lub przeprowadzenia zakrojonych na szeroką skalę ankiet. Takie testy przeprowadza się zwykle tylko raz w cyklu wytwarzania.

Wiele innych rodzajów testów niefunkcjonalnych (np. testy bezpieczeństwa, testy wydajności) wymaga środowiska zbliżonego do produkcyjnego. Ponieważ koszt takiego środowiska zwykle jest

wysoki, jedyną praktycznie dostępną możliwością może być użycie samego środowiska produkcyjnego. Harmonogram wykonywania takich testów musi być starannie zaplanowany. Często takie testy mogą być wykonywane tylko w określonym czasie (np. nocą).

Do wszystkich testów związanych z wydajnością (np. osiągi, obciążenie) powinny być zaplanowane odpowiednie komputery i odpowiednia przepustowość łącza. Potrzeby zależą głównie od liczby wirtualnych użytkowników, których trzeba zasymulować oraz od wielkości obciążenia, które oni mogą generować. Jeśli nie weźmie się tego pod uwagę, mogą pojawić się niereprezentatywne wyniki testów wydajności.

3.11.4.4 Zagadnienia organizacyjne

Testy niefunkcjonalne obejmują pomiary działania wielu komponentów w całym systemie (na przykład serwerów, baz danych, sieci). Jeśli komponenty te są rozproszone w wielu miejscach i w różnych organizacjach, zaplanowanie i koordynacja testów mogą wymagać wiele pracy. Przykładowo, niektóre komponenty programowe mogą być dostępne tylko w określonych porach dnia lub roku, albo organizacje mogą zezwalać na testowanie tylko przez określoną liczbę dni. Brak dostępności komponentów systemu i personelu z innych organizacji może poważnie zaburzyć harmonogram zaplanowanych testów.

3.11.4.5 Zagadnienia dotyczące komunikacji

Możliwość zdefiniowania i wykonania pewnych rodzajów testów niefunkcjonalnych (w szczególności testów efektywności) zależy od możliwości modyfikowania pewnych protokołów komunikacyjnych na potrzeby testowe. Podczas planowania należy to uwzględnić, np. wziąć pod uwagę potrzebę kompatybilności komunikacyjnej narzędzi.

3.11.4.6 Zagadnienia bezpieczeństwa danych

Szczególne zabezpieczenia wbudowane w system muszą być uwzględnione już w fazie planowania testów, aby upewnić się, że wykonanie wszystkich czynności testowych jest w ogóle możliwe. Na przykład, szyfrowanie danych może utrudniać tworzenie danych testowych i weryfikację wyników.

Prawa o ochronie danych mogą spowodować konieczność posługiwania się wirtualnymi użytkownikami, stworzonymi na podstawie danych produkcyjnych. Zabezpieczenie anonimowości danych testowych nie jest zadaniem łatwym i musi być zaplanowane i wykonane podczas implementacji testów.

4. Techniki Testowania

Terminologia

BS 7925-2, analiza wartości brzegowych, testowanie rozgałęzień, tworzenie grafów przyczynowo-skutkowych, metoda drzewa klasyfikacji, testowanie warunków, pokrycie warunków znaczących, analiza przepływu sterowania, ścieżka D-D, analiza przepływu danych, testowanie w oparciu o tablicę decyzyjną, testowanie decyzji, technika oparta na defektach, taksonomia defektów, analiza dynamiczna, zgadywanie błędów, podział na klasy równoważności, testowanie eksploracyjne, technika oparta na doświadczeniu, LSKiS, wyciek pamięci, testowanie wielokrotnych warunków, testowanie par, testowanie ścieżek, testowanie oparte na wymaganiach, atak na oprogramowanie, technika oparta na specyfikacji, analiza statyczna, testowanie instrukcji, testowanie przejść pomiędzy stanami, technika oparta o strukturę, statut testu, testowanie w oparciu o przypadki użycia, dziki wskaźnik.

4.1 Wprowadzenie

Techniki projektowania testów omawiane w tym rozdziale należą do następujących kategorii:

- oparte na specyfikacji (oparte na zachowaniu, czarnoskrzynkowe)
- oparte o strukturę (białoskrzynkowe)
- oparte na defektach
- oparte na doświadczeniu

Wymienione techniki uzupełniają się i mogą być wykorzystywane odpowiednio we wszystkich czynnościach testowych, bez względu na poziom przeprowadzanych testów. Mimo, że każda z nich może być stosowana zarówno przez analityka testów, jak i przez technicznego analityka testów, bazując na najczęstszych przypadkach, na potrzeby tego podręcznika przyjęto następujący podział:

- oparte na specyfikacji -> analityk testów i techniczny analityk testów
- oparte o strukturę -> techniczny analityk testów
- oparte na doświadczeniu -> analityk testów i techniczny analityk testów
- oparte na defektach -> analityk testów i techniczny analityk testów

Dodatkowo w stosunku do wymienionych obszarów, w rozdziale omówione zostaną również inne techniki, takie jak ataki, analiza statyczna i analiza dynamiczna. Wymienione techniki są zazwyczaj stosowane przez technicznych analityków testów.

Należy zauważyć, że techniki oparte na specyfikacji mogą być zarówno funkcjonalne, jak i niefunkcjonalne. Niefunkcjonalne techniki są omówione w następnym rozdziale.

4.2 Techniki oparte na specyfikacji

Techniki oparte na specyfikacji są metodą tworzenia i dobierania warunków testowych lub przypadków testowych, opartą o analizę dokumentacji będącej podstawą testów modułu lub systemu, bez wnikania w jego wewnętrzną strukturę.

Podstawowe cechy technik opartych na specyfikacji:

- Do definiowania otwartych zadań/problemów, specyfikowania aplikacji lub jej modułów, stosowane są modele (formalne, jak i nieformalne).
- Z takich modeli przypadki testowe mogą być tworzone w sposób systematyczny.

Niektóre techniki dostarczają kryteria pokrycia, które mogą być stosowane jako metryki dla zadań projektowania testów. Osiągnięcie pełnego pokrycia nie oznacza jednak, że zestaw testów jest kompletny, ale raczej, że model nie proponuje więcej przydatnych testów opartych na tej technice.

Testy oparte na specyfikacji często są testami opartymi na wymaganiach. Ponieważ specyfikacja wymagań powinna definiować, jak system ma działać, szczególnie w zakresie funkcjonalnym, tworzenie testów na podstawie wymagań często staje się częścią testowania działania systemu. Techniki omawiane wcześniej w tej sekcji, mogą być zastosowane bezpośrednio do stworzenia modelu działania systemu na podstawie tekstowych opisów wymagań lub diagramów zawartych w specyfikacji wymagań, a w kolejnym kroku do stworzenia samych testów, jak to opisano wcześniej.

W tym syllabusie rozważane są następujące techniki oparte na specyfikacji:

Technika	**Opis**	**Kryteria pokrycia**
Podział na klasy równoważności	Opis techniki znajduje się w syllabusie poziomu podstawowego w rozdziale 4.3.1.	Liczba pokrytych klas/ całkowita liczba klas
Analiza wartości brzegowych (AWB)	Opis techniki znajduje się w syllabusie poziomu podstawowego w rozdziale 4.3.2. Należy zwrócić uwagę, że AWB może być zastosowana z użyciem 2 lub 3 wartości. Decyzja o tym, którą z nich zastosować powinna być podjęta w oparciu o analizę ryzyka.	Liczba pojedynczych wartości brzegowych pokrytych testami / całkowita liczba wartości brzegowych
Testowanie w oparciu o tablicę decyzyjną oraz tworzenie grafów	Opis techniki znajduje się w syllabusie poziomu podstawowego w rozdziale 4.3.3. W oparciu o tablicę	Liczba pokrytych kombinacji warunków / maksymalna liczba

przyczynowo-skutkowych	decyzyjną testowana jest każda kombinacja warunków, zależności i ograniczeń. Jako uzupełnienie do tablic decyzyjnych może być stosowana graficzna technika wykorzystująca logiczną notację, zwana techniką tworzenia grafów przyczynowo-skutkowych. Należy zauważyć, że zamiast testowania każdej kombinacji warunków, można również użyć tablic uproszczonych. Decyzja o tym, czy zastosować pełną tablicę decyzyjną czy tablicę uproszczoną, powinna być oparta na analizie ryzyka [Copeland03].	kombinacji warunków
Testowanie przejść pomiędzy stanami	Opis techniki znajduje się w syllabusie poziomu podstawowego w rozdziale 4.3.4.	Dla pojedynczych przejść, metryką pokrycia jest procent wszystkich możliwych przejść wykonanych podczas testu. Jest to pokrycie znane jako 0-przełączeń. Dla *n* przejść miarą pokrycia jest procent wszystkich możliwych sekwencji *n* przejść, wykonanych podczas testu. Jest to pokrycie (n-1) przełączeń.
Metoda drzewa klasyfikacji, tablice ortogonalne i tablica wszystkich par	Identyfikowane są współczynniki lub zmienne wraz z opcjami i wartościami, jakie mogą przyjmować. Następnie identyfikowane są pary, trójki lub nawet kombinacje wyższego rzędu tych opcji lub wartości [Copeland03]. Metoda drzewa klafikacji wykorzystuje graficzną notację aby pokazać warunki (podzbiory) testów oraz kombinacji adresowanych przez przypadki testowe [Grochtmann94].	Zależą od zastosowanej techniki, np. kryteria pokrycia dla testowania par są inne niż te dla drzew klasyfikacji.

Testowanie w oparciu o przypadki użycia	Opis techniki znajduje się w syllabusie poziomu podstawowego w rozdziale 4.3.5.	Nie stosuje się żadnych formalnych kryteriów.

Czasami podczas tworzenia testów łączone są różne techniki. Na przykład, warunki zidentyfikowane za pomocą tablicy decyzyjnej mogą okazać się użyteczne przy podziale na klasy równoważności do odkrycia wielu sposobów na spełnienie danego warunku. Testy pokrywałyby w takim przypadku nie tylko każdą kombinację warunków, ale także wygenerowane zostałyby dodatkowe testy do pokrycia klas równoważności, dla warunków tego wymagających.

4.3 Techniki oparte o strukturę

Technika projektowania testów oparta o strukturę, zwana także białoskrzynkową lub techniką testowania na podstawie kodu, jest metodą, w której kod, dane, architektura lub przepływ sterowania są wykorzystywane jako podstawa do projektowania testów, a przypadki testowe są tworzone systematycznie na podstawie struktury. Technika determinuje elementy struktury, które mają być uwzględniane. Dostarcza ona również kryteria pokrycia, które określają, kiedy tworzenie przypadków testowych ma się zakończyć. Osiągnięcie tych kryteriów nie oznacza jednak, że uzyskany zestaw testów jest kompletny, ale raczej, że dalsza analiza struktury nie dostarczy więcej przydatnych testów. Każde z kryteriów powinno być mierzalne, a jednocześnie powiązane z celem zdefiniowanym na poziomie projektu lub całej firmy.

Podstawowe cechy technik opartych o strukturę:

- Do tworzenia przypadków testowych wykorzystywane są informacje o budowie oprogramowania (tych informacji mogą dostarczać np. kod i projekt).
- Mierzone jest pokrycie oprogramowania przypadkami testowymi, a zwiększenie pokrycia testowego można uzyskać poprzez systematyczne tworzenie kolejnych przypadków testowych.

W tym syllabusie rozważane są następujące techniki oparte o strukturę:

Technika	**Opis**	**Kryteria pokrycia**
Testowanie instrukcji	Identyfikowane są instrukcje wykonywalne (komentarze i spacje są pomijane).	liczba pokrytych instrukcji / liczba wszystkich instrukcji
Testowanie decyzji	Identyfikowane są instrukcje warunkowe, takie jak *if/else*, *switch/case*, *for* i *while.*	liczba pokrytych wyników decyzji / liczba wszystkich możliwych wyników decyzji

Testowanie rozgałęzień	Identyfikowane są rozgałęzienia, takie jak *if/else*, *switch/case*, *for* i *while.* Należy zwrócić uwagę, że testowanie rozgałęzień i testowanie decyzji jest identyczne przy 100%-owym pokryciu, ale może się różnić w przypadku niższych poziomów pokrycia.	liczba pokrytych rozgałęzień / liczba wszystkich rozgałęzień
Testowanie warunków	Identyfikowane są warunki *true/false* oraz *case*.	liczba pokrytych wartości operatorów boolowskich / liczba wszystkich wartości operatorów boolowskich
Testowanie wielokrotnych warunków	Identyfikowane są wszystkie możliwe kombinacje warunków *true/false*.	liczba pokrytych kombinacji wartości operatorów boolowskich / liczba wszystkich kombinacji wartości operatorów boolowskich
Pokrycie warunków znaczących	Identyfikowane są wszystkie możliwe kombinacje warunków *true/false*, które mogą wpływać na decyzje (rozgałęzienia).	liczba pokrytych operatorów boolowskich, które niezależnie wpływają na wynik decyzji / liczba wszystkich operatorów boolowskich

LSKiS (testowanie pętli)	Identyfikowane są wszystkie możliwe warunki, które wpływają na iteracje w pętli. Liniowa Sekwencja Kodu i Skok (LSKiS) jest używana do testowania fragmentu kodu (liniowej sekwencji wykonywalnego kodu), który rozpoczyna się na początku wybranego bloku sterowania i kończy skokiem do innego bloku sterowania programu. Takie fragmenty kodu znane są również jako ścieżki D-D. Opisywana technika jest stosowana do definiowania przypadków i danych testowych, które mają na celu sprawdzenie konkretnego, wybranego przepływu sterowania. Przy projektowaniu takich testów wymagane jest posiadanie dostępu do modelu kodu źródłowego, w którym zdefiniowano skoki w przepływie sterowania. LSKiS mogą być wykorzystane jako podstawa do pomiaru pokrycia kodu.	liczba pokrytych LSKiS / liczba wszystkich LSKiS
Testowanie ścieżek	Identyfikowane są unikalne sekwencje instrukcji (ścieżki).	liczba pokrytych ścieżek / liczba wszystkich ścieżek

Jednym z powszechnych zastosowań kryteriów pokrycia strukturalnego jest pomiar kompletności zestawu testów zaprojektowanych przy pomocy technik opartych na specyfikacji i/lub technik opartych na doświadczeniu. Pokrycie strukturalne osiągnięte przy pomocy takich testów jest rejestrowane przez narzędzia testowe zwane narzędziami pokrycia kodu. Jeśli wykryte zostaną poważne luki w pokryciu strukturalnym (lub poziom pokrycia wymagany przez stosowne standardy nie został osiągnięty), do pokrycia tych luk są dodawane testy zaprojektowane w oparciu o techniki strukturalne i/lub inne.

Analiza pokrycia

Chcąc stwierdzić, czy dany poziom pokrycia kodu został osiągnięty, czy też nie, można przeprowadzić testy dynamiczne, stosując techniki oparte o strukturę lub inne. W przypadku systemów krytycznych, dla których określone pokrycie musi być obowiązkowo osiągnięte, ma to na celu udowodnienie osiągniętego pokrycia (zobacz także rozdział 1.3.2 Systemy krytyczne ze względu na bezpieczeństwo).

Wyniki mogą wskazywać, że niektóre fragmenty kodu nie zostały sprawdzone, co powinno prowadzić do zdefiniowania dodatkowych przypadków testowych, które te sekcje sprawdzą.

4.4 Techniki oparte na defektach i techniki oparte na doświadczeniu

4.4.1 Techniki oparte na defektach

Technika projektowania testów oparta na defektach jest techniką, w której typ szukanego defektu jest podstawą do projektowania przypadków testowych, a testy tworzone są systematycznie w oparciu o wiedzę o typie defektu.

Technika ta dostarcza również kryteria pokrycia, które są stosowane do określenia, kiedy można zakończyć tworzenie testów. W praktyce, kryteria pokrycia dla technik opartych na defektach bywają mniej systematyczne niż kryteria dla technik opartych na zachowaniu i technik opartych na strukturze. Wynika to z faktu, że dla tej techniki zasady pokrycia są ogólne, a decyzja o tym, co stanowi limit użytecznego pokrycia jest niejednoznaczna i może się zmieniać w trakcie projektowania lub wykonania testów. Osiągnięcie pełnego pokrycia nie oznacza, że zestaw testów jest kompletny, ale raczej że analiza rozważanych defektów nie sugeruje już więcej przydatnych testów opartych na tej technice.

W tym syllabusie rozważana jest następująca technika oparta na defektach:

Technika	**Opis**	**Kryteria pokrycia**
Taksonomie (kategorie i listy potencjalnych defektów)	Tester wybiera potencjalny problem do analizy, korzystając z listy, na której mogą znajdować się przyczyny, defekty i awarie. Taksonomie defektów zawierają najpowszechniejsze defekty w testowanej aplikacji. Lista używana jest do projektowania przypadków testowych.	Stosowne dane o defektach i typach defektów

4.4.2 Techniki oparte na doświadczeniu

Istnieją również inne techniki projektowania testów, które uwzględniają historię defektów, ale nie koniecznie charakteryzują się systematycznymi kryteriami pokrycia. Tego rodzaju techniki zostały skategoryzowane jako techniki oparte na doświadczeniu.

W testach opartych na doświadczeniu wykorzystywane są umiejętności i intuicja testerów oraz ich doświadczenie w testowaniu podobnych aplikacji lub technologii. Takie testy są efektywne w znajdywaniu defektów ale nie sprawdzają się tak dobrze, jak inne techniki do osiągania określonych poziomów pokrycia, ani do tworzenia reużywalnych procedur testowych.

Stosując dynamiczne i heurystyczne podejścia, testerzy zazwyczaj wykonują testy oparte na doświadczeniu a testowanie staje się bardziej reaktywne na zdarzenia niż w przypadku podejść planowanych. Ponadto wykonanie i ocena wyników testów są wykonywane równocześnie. Niektóre bardziej strukturalne podejścia do testów opartych na doświadczeniu nie są do końca dynamiczne, np. testy nie są w całości tworzone w momencie ich wykonywania.

Należy zauważyć, iż mimo że w poniższej tabeli zaprezentowano kilka pomysłów dotyczących pokrycia, techniki oparte na doświadczeniu nie posiadają żadnych formalnych kryteriów pokrycia.

W tym syllabusie rozważane są następujące techniki oparte na doświadczeniu:

Technika	**Opis**	**Kryteria pokrycia**
Zgadywanie błędów	Tester wykorzystuje swoje doświadczenie do zgadywania potencjalnych błędów i sam wybiera metody odkrywania defektów. Zgadywanie błędów przydaje się także podczas analizy ryzyka identyfikowania potencjalnych trybów awarii [Myers97].	Gdy wykorzystywana jest taksonomia, pokrycie odpowiednich awarii oraz rodzajów defektów. Bez taksonomii, ograniczeniami dla pokrycia są doświadczenie testera oraz czas dostępny na wykonanie testów.

Testowanie w oparciu o listy kontrolne	Doświadczony tester stosuje: • ogólną listę, z wyszczególnionymi elementami na które należy zwrócić uwagę i które powinny zostać sprawdzone, bądź zapamiętane, lub • zestaw reguł lub kryteriów, w odniesieniu do których produkt będzie weryfikowany. Listy kontrolne są budowane w oparciu o standardy, doświadczenie i inne stosowne uwarunkowania. Przykładem testu w oparciu o listę kontrolną może być wykorzystanie listy kontrolnej zawierającej standardy dotyczące interfejsu użytkownika jako podstawy do przetestowania aplikacji.	Każdy element z listy kontrolnej został pokryty testami.
Testowanie eksploracyjne	Tester równocześnie uczy się o produkcie i jego defektach, planuje prace testowe, projektuje i wykonuje testy oraz raportuje wyniki. Dobre testy eksploracyjne są zaplanowane, interaktywne i twórcze. Tester dynamicznie dostosowuje cele testowania w trakcie wykonania testów i przygotowuje tylko pobieżną dokumentację.	Można tworzyć statuty testów, które specyfikują zadania, cele oraz produkty końcowe. Dodatkowa sesja testów eksploracyjnych może pomóc nam zidentyfikować, co chcemy osiągnąć, na czym należy się skupić, co wchodzi w zakres a co jest poza zakresem testów oraz jakie zasoby powinny zostać zaangażowane. Można ponadto wyodrębnić zestaw reguł dotyczących defektów i jakości.

Ataki	Tester dokonuje oceny jakości systemu poprzez próby wywołania określonych awarii. Podstawą ataków na oprogramowanie jest wykorzystanie interakcji testowanej aplikacji ze środowiskiem, w którym jest uruchamiana [Whittaker03]. Zalicza się do tego interfejs użytkownika, system operacyjny z jądrem, API oraz system plików. Interakcje oparte są na precyzyjnej wymianie danych. Każde zakłócenie w jednej (lub wielu) z tych interakcji może być przyczyną awarii.	Różne interfejsy testowanej aplikacji. Głównymi interfejsami są: interfejs użytkownika, interfejs systemu operacyjnego, jądro, API oraz system plików.

W technikach opartych na defektach i na doświadczeniu stosuje się wiedzę o defektach oraz inne doświadczenie w celu zwiększenia poziomu wykrywalności defektów. Możemy mieć do czynienia z bardzo szybkimi testami, w przypadku których tester nie ma formalnie zaplanowanych zadań do wykonania, poprzez zaplanowane sesje aż po sesje, dla których przygotowano skrypty testowe. Opisywane testy są prawie zawsze użyteczne, ale mają szczególną wartość w następujących okolicznościach, gdy:

- brak specyfikacji,
- dokumentacja testowanego systemu jest uboga,
- czas na zaprojektowanie i stworzenie procedur testowych jest niewystarczający,
- testerzy są doświadczeni w danej dziedzinie i/lub technologii,
- poszukiwane jest uzupełnienie dla testowania skryptowego,
- analizowane są awarie operacyjne.

Techniki oparte na defektach i na doświadczeniu stanowią również dobre uzupełnienie dla technik opartych na zachowaniu i strukturze, adresując słabe punkty tych drugich.

4.5 Analiza statyczna

Analiza statyczna wiąże się z testowaniem bez uruchamiania testowanej aplikacji i może dotyczyć kodu lub architektury systemu.

4.5.1 Analiza statyczna kodu

Pod pojęciem analizy statycznej mieści się wszelkiego rodzaju testowanie lub sprawdzanie, które może być zrealizowane bez wykonywania kodu (uruchamiania) aplikacji. Istnieje kilka technik analizy statycznej, a zostały one omówione w niniejszym rozdziale.

4.5.1.1 Analiza przepływu sterowania

Analiza przepływu sterowania dostarcza informacji o logicznych punktach decyzyjnych w kodzie programu i złożoności ich struktury. Analiza przepływu sterowania została opisana w syllabusie poziomu podstawowego oraz w [Beizer95].

4.5.1.2 Analiza przepływu danych

Analiza przepływu danych jest techniką strukturalną, która testuje ścieżki pomiędzy ustawieniem zmiennej a jej późniejszym użyciem. Takie ścieżki są nazywane parami definicja-użycie lub też parami ustawienie-użycie. W tej metodzie generowane są zestawy testów do osiągnięcia 100% pokrycia (tam gdzie to możliwe) dla każdej z takich par.

Technika, jakkolwiek zwana analizą przepływu danych, uwzględnia również przepływ sterowania w testowanej aplikacji, śledząc ustawienie i użycie każdej ze zmiennych (co może wymagać podążania za przepływem sterowania programu). Zobacz również [Beizer95].

4.5.1.3 Testowanie zgodności ze standardami programowania

Podczas analizy statycznej weryfikowana może być również zgodność ze standardami kodowania. Standardy kodowania dotyczą zarówno aspektów architekturalnych, jak i zaleceń (lub zakazów) używania pewnych metod programistycznych.

Zgodność ze standardami czyni oprogramowanie łatwiej utrzymywalnym i testowalnym. Podczas testów statycznych mogą być również weryfikowane specyficzne wymagania właściwe dla danego języka programowania.

4.5.1.4 Generowanie metryk kodu

Podczas analizy statycznej mogą być generowane metryki kodu, których znajomość wspomaga utrzymanie kodu oraz podnosi jego poziom niezawodności. Metryki, o których tutaj mowa to m.in.:

- Złożoność cyklomatyczna
- Wielkość kodu
- Częstotliwość występowania komentarzy
- Liczba zagnieżdżonych poziomów
- Liczba wywołań funkcji

4.5.2 Analiza statyczna architektury

4.5.2.1 Analiza statyczna strony webowej

Również architektura strony webowej może być sprawdzana przy wykorzystaniu narzędzi do analizy statycznej. W tym przypadku celem jest sprawdzenie, czy struktura drzewiasta strony jest dobrze wyważona lub czy występuje brak równowagi, który może prowadzić do:

- utrudnienia czynności testowych,
- zwiększonej pracochłonności utrzymania,

- trudniejszej dla użytkownika nawigacji.

Niektóre specjalistyczne narzędzia testowe, zawierające mechanizm automatycznej analizy stron (ang. *web spider engine*), w oparciu o analizę statyczną mogą również dostarczać informacji o rozmiarze stron, o czasie ich ładowania a także o jej dostępności lub jej braku (np. błąd http 404). Takie informacje mogą być przydatne zarówno dla programisty, projektanta strony, jak i dla testera.

4.5.2.2 Grafy wywołań

Grafy wywołań mają szereg zastosowań:

- Projektowanie testów wywołujących określony moduł.
- Ustalanie liczby lokalizacji wewnątrz programu, z których wywoływany jest dany moduł.
- Sugerowanie kolejności integracji (integracja parami oraz integracja modułów sąsiednich [Jorgensen02]).
- Ocena struktury całego kodu oraz jego architektury.

Informacje o wywoływanych modułach można również uzyskać podczas analizy dynamicznej. Uzyskana w ten sposób informacja odnosi się do liczby wywołań modułu podczas wykonania testu. Poprzez połączenie informacji uzyskanej z grafów wywołań podczas analizy statycznej z informacją uzyskaną w drodze analizy dynamicznej, tester może się skupić na modułach, które są wywoływane często i/lub wielokrotnie. Dodatkowo, w sytuacji, gdy moduły, o których tutaj mowa są złożone (zobacz 1.3.2.2 Systemy krytyczne ze względu na bezpieczeństwo i złożoność), stają się one pierwszymi kandydatami do szczegółowych testów.

4.6 Analiza dynamiczna

Analiza dynamiczna polega na analizie aplikacji podczas jej działania. Często wymaga to instrumentalizacji, dodanej do kodu ręcznie lub automatycznie.

4.6.1 Wprowadzenie

Defekty, których nie daje się natychmiast odtworzyć (powtórzyć), mogą w konsekwencji prowadzić do znacznego zwiększenia nakładów przeznaczonych na testy, jak również uniemożliwiać wydanie wersji oprogramowania lub efektywne korzystanie z oprogramowania. Przyczyną takich defektów mogą być wycieki pamięci, niewłaściwe użycie wskaźników lub inne nieprawidłowości (np. uszkodzenie stosu systemowego) [Kaner02]. Defekty tego typu mogą prowadzić do stopniowego spadku wydajności lub nawet do całkowitego zawieszenia systemu. Z tego względu, strategie testowe muszą uwzględniać ryzyka związane z takimi defektami oraz stosować (tam gdzie to możliwe) analizę dynamiczną do zapobiegania tym ryzykom (zazwyczaj przy pomocy odpowiednich narzędzi).

Analiza dynamiczna jest przeprowadzana w trakcie działania aplikacji i może być stosowana do:

- Zapobiegania awariom, poprzez wykrywanie dzikich wskaźników (ang. *wild pointers*) i wycieków pamięci (ang. *memory leaks*).
- Analizy awarii, które trudno jest odtworzyć
- Oceny działania sieci

- Podnoszenia wydajności systemu poprzez dostarczanie informacji o zachowaniu systemu podczas jego działania

Analiza dynamiczna może zostać przeprowadzona na każdym poziomie testów, ale najbardziej użyteczna jest podczas testów modułowych i integracyjnych. Do określenia celów analizy dynamicznej, a w szczególności do analizy jej wyników, niezbędne są umiejętności techniczne i systemowe.

4.6.2 Wykrywanie wycieków pamięci

Wyciek pamięci (ang. *memory* leak) występuje wtedy, kiedy pamięć (RAM) dostępna dla programu, została przez ten program zaalokowana ale nie została potem zwolniona ze względu na błędy programistyczne. Program traci możliwość dostępu do pamięci i w efekcie może to prowadzić do braku wolnej pamięci do działania.

Wycieki pamięci powodują problemy, które narastają z czasem i nie zawsze mogą być od razu uwidocznione. Przeoczenie problemu może mieć miejsce w przypadkach, gdy oprogramowanie zostało niedawno zainstalowane lub też restartowano system, co często zdarza się podczas testów. Z tych względów, negatywne skutki wycieków pamięci często zostają dostrzeżone dopiero po wdrożeniu produkcyjnym.

Symptomem wycieku pamięci jest powolne, stopniowe pogarszanie się czasów odpowiedzi systemu, które może ostatecznie zakończyć się awarią systemu. Mimo iż takie awarie mogą być rozwiązywane poprzez restartowanie systemu, takie działanie nie jest praktyczne a czasem bywa nawet niemożliwe.

Istnieją narzędzia, które potrafią identyfikować obszary kodu, w których występują wycieki pamięci, dzięki czemu można je usunąć. Można użyć również prostych monitorów pamięci do obserwacji, czy dostępna pamięć zmniejsza się w czasie. W razie wystąpienia tego zjawiska wymagana będzie dodatkowa analiza.

Istnieją również inne rodzaje wycieków, które powinny być wzięte pod uwagę. Do przykładów można zaliczyć wycieki uchwytów plików (ang. *file handles*), semaforów i pul połączeń do różnych zasobów.

4.6.3 Wykrywanie dzikich wskaźników

Dzikie wskaźniki w programie (ang. *wild pointers*) to wskaźniki, których z jakiegoś powodu nie da się użyć. Na przykład mogły one „zgubić” obiekt lub funkcję, na które powinny wskazywać, lub też nie wskazują na ten obszar pamięci, na który powinny (np. poza granice zaalokowanej tabeli). Odwołania do dzikich wskaźników mogą mieć różne konsekwencje:

1. Program może zachowywać się zgodnie z oczekiwaniami. Taka sytuacja może się zdarzyć, jeśli dziki wskaźnik odwołuje się do obszaru pamięci, który jest „wolny” i nie jest aktualnie wykorzystywany przez program.

2. Program może się zawiesić. W takim przypadku dziki wskaźnik mógł użyć obszaru pamięci, który jest krytyczny dla działania programu (np. obszar pamięci należący do systemu operacyjnego).

3. Program nie funkcjonuje prawidłowo, ponieważ nie może uzyskać dostępu do obiektów niezbędnych do jego działania. W takich warunkach program może wciąż funkcjonować, ale będą pojawiać się komunikaty o błędach.

4. Pod adresem, do którego odwołuje się dziki wskaźnik, mogą zostać nadpisane dane a następnie te niewłaściwe wartości mogą zostać wykorzystane przez program.

Należy pamiętać, że jakiekolwiek zmiany w zakresie wykorzystania pamięci wprowadzone do programu (np. w nowej wersji aplikacji) mogą być potencjalną przyczyną wymienionych wyżej objawów. Szczególnie krytyczne jest to w sytuacjach, gdy początkowo program działa zgodnie z oczekiwaniami mimo istnienia dzikich wskaźników, a potem zupełnie nieoczekiwanie zawiesza się (być może nawet podczas eksploatacji produkcyjnej) tuż po wgraniu nowej wersji. Ważne jest, by zwrócić uwagę, że takie awarie są symptomami ukrytego błędu (np. dzikiego wskaźnika) a nie błędem samym w sobie (patrz też [Kaner02], „Lesson 74”).

Narzędzia potrafią wykrywać dzikie wskaźniki w momencie, gdy są one używane przez program, niezależnie od konsekwencji, jakie wywołują w jego działaniu.

4.6.4 Analiza wydajności

Celem analizy dynamicznej może być również ocena wydajności programu. W wykrywaniu „wąskich gardeł” wydajnościowych i generowaniu szerokiej gamy metryk wydajnościowych bardzo pomocne są odpowiednie narzędzia. Dostarczane przez nie dane mogą być wykorzystane przez programistów do strojenia systemu. Nazywa się to również „profilowaniem wydajności”.

5. Testowanie właściwości oprogramowania

Terminologia

testowanie dostępności, testowanie dokładności, testowanie efektywności, ocena heurystyczna, testowanie współdziałania[7], testowanie pielęgnowalności, produkcyjne testy akceptacyjne (ang. *production acceptance testing* lub *operational acceptance testing – OAT*), profil operacyjny, testy przenaszalności, testowanie odtwarzalności, model wzrostu niezawodności, testowanie niezawodności, testowanie zabezpieczeń, testowanie odpowiedniości, kwestionariusz oceny użyteczności oprogramowania (ang. *Software Usability Measurement Invertory - SUMI*), testowanie użyteczności

5.1 Wstęp

Podczas, gdy poprzedni rozdział opisywał różne techniki dostępne testerowi, ten przedstawia ich zastosowanie do oceny podstawowych atrybutów używanych w celu opisu jakości oprogramowania lub systemów.

W syllabusie każdy z atrybutów jakościowych podlegających ocenie analityka testów oraz technicznego analityka testów został przedstawiony w osobnej sekcji. Podstawą opisu atrybutów jakościowych w niniejszym dokumencie jest opis zawarty w normie ISO 9126. Podstawowym celem kształcenia dla każdego z trzech modułów jest zrozumienie różnych atrybutów jakościowych. W zależności od konkretnego atrybutu jakościowego, jego znajomość jest pogłębiana w module analityka testów lub technicznego analityka testów, w celu identyfikacji typowych obszarów ryzyka, rozwoju odpowiednich strategii testowania i przygotowania przypadków testowych.

5.2 Atrybuty jakościowe do testowania dziedzinowego

Testowanie funkcjonalne skupia się na tym, „co" produkt robi. Podstawą testu dla testów funkcjonalnych jest najczęściej dokument wymagań lub specyfikacja, konkretna wiedza dziedzinowa lub domyślna potrzeba (ang. *implied need*). Testy funkcjonalne różnią się od siebie w zależności od poziomu lub fazy testów, w której są wykonywane. Na przykład, testy funkcjonalne przeprowadzane podczas testów integracyjnych będą sprawdzały funkcjonalność komunikujących się ze sobą modułów, które realizują jedną określoną funkcję. Na poziomie testów systemowych, testy funkcjonalne będą miały na celu testowanie funkcjonalności całości aplikacji. Dla systemów złożonych testy funkcjonalne będą się skupiać na testach pełnych procesów biznesowych (ang. *end-to-end testing*), przechodząc przez zintegrowane systemy.

Podczas testów funkcjonalnych stosowany jest szeroki zakres technik testowych (patrz rozdział 4). Testy funkcjonalne mogą być wykonywane przez wyznaczonego testera, eksperta dziedzinowego lub programistę (zwykle na poziomie modułowym).

[7] Wcześniej tłumaczono jako „międzyoperacyjności" [przypis SJSI].

Podczas tego typu testów bierze się pod uwagę następujące atrybuty:

- dokładność
- odpowiedniość
- współdziałanie
- zabezpieczenia funkcjonalne
- użyteczność
- dostępność

5.2.1 Testowanie dokładności (ang. *accuracy testing*)

Dokładność funkcjonalna dotyczy testowania zgodności aplikacji z wyspecyfikowanymi lub domyślnymi wymaganiami; może dotyczyć również testowania dokładności obliczeń. W testowaniu dokładności wykorzystuje się wiele technik z rozdziału 4.

5.2.2 Testowanie odpowiedniości (ang. *suitability testing*)

Testowanie odpowiedniości polega na ocenie i walidacji, czy zbiór funkcji jest odpowiedni do spełnienia konkretnych, założonych zadań. Ten rodzaj testów może bazować na przypadkach użycia lub procedurach.

5.2.3 Testowanie współdziałania (ang. *interoperability testing*)

Celem testu współdziałania jest sprawdzenie, czy dana aplikacja może funkcjonować prawidłowo we wszystkich docelowych środowiskach (sprzęt, oprogramowanie, warstwa integracyjna, system operacyjny itd.). Specyfikowanie testów współdziałania wymaga zidentyfikowania kombinacji docelowych środowisk, jak również skonfigurowania oraz udostępnienia ich zespołowi testowemu. Na tych środowiskach wykonuje się następnie wybrane z zestawu testów funkcjonalnych przypadki testowe, których celem jest sprawdzenie różnych modułów systemu.

Współdziałanie odnosi się do tego, jak różne systemy oprogramowania współpracują ze sobą. Oprogramowanie charakteryzujące się wysokim stopniem współdziałania daje się łatwo, bez większych zmian, zintegrować z wieloma innymi systemami. Liczba zmian oraz pracochłonność ich wykonania mogą być użyte jako miara stopnia współdziałania.

Testowanie współdziałania oprogramowania może skupiać się na przykład na następujących cechach:

- wykorzystanie przez oprogramowanie szeroko stosowanych standardów komunikacyjnych, takich jak XML
- możliwość automatycznego wykrywania wymagań komunikacyjnych systemów, z którymi oprogramowanie może wchodzić w interakcję oraz dostosowywanie się do nich

Testowanie współdziałania może być szczególnie istotne dla:

- organizacji rozwijających oprogramowanie i narzędzi „z półki” (ang. *commercial off the shelf - COTS*)
- organizacji rozwijających systemy systemów

Ta forma testowania jest obecna głównie w testowaniu integracji systemów.

5.2.4 Funkcjonalne testowanie zabezpieczeń (ang. *functional security testing*)

Funkcjonalne testowanie zabezpieczeń (testowanie penetracyjne) skupia się na zdolności oprogramowania do zapobiegania nieuprawnionemu dostępowi do funkcji oraz danych - dostępowi, przypadkowemu i/lub celowemu. Testy te zawierają sprawdzenie uprawnień użytkowników, dostępów i przywilejów. Informacje te powinny być dostępne w specyfikacji systemu. Testowanie zabezpieczeń obejmuje również wiele aspektów bardziej przydatnych technicznym analitykom testów, i są one omówione w podrozdziale 5.3 poniżej.

5.2.5 Testowanie użyteczności (ang. *usability testing*)

Bardzo ważne jest zrozumienie, dlaczego użytkownicy mogą mieć problemy z używaniem danego systemu. Żeby to osiągnąć, należy wziąć pod uwagę, że pojęcie "użytkownik" może dotyczyć szerokiego wachlarza ludzi, poczynając od ekspertów IT, a kończąc na dzieciach lub osobach niepełnosprawnych.

Niektóre instytucje narodowe (np. the British Royal National Institute for the Blind[8]) postulują uczynienie stron WWW dostępnymi dla osób niepełnosprawnych, niewidomych, słabowidzących, niepełnosprawnych ruchowo, głuchych oraz ludzi z innymi zaburzeniami poznawczymi. Potwierdzenie, że aplikacje i strony WWW są użyteczne/używalne dla powyższych użytkowników, podniesie również poziom użyteczności dla wszystkich pozostałych.

Testowanie użyteczności mierzy odpowiedniość oprogramowania dla jego użytkowników i koncentruje się na badaniu następujących charakterystyk, przy pomocy których użytkownicy mogą osiągać określone cele w konkretnych środowiskach lub kontekstach użycia:

- efektywność - zdolność oprogramowania do umożliwienia użytkownikom dokładnej i kompletnej realizacji założonych celów w określonym kontekście użycia
- wydajność - zdolność oprogramowania do umożliwienia użytkownikom osiągnięcia odpowiedniej efektywności w określonym kontekście użycia przy wykorzystaniu właściwej ilości zasobów
- satysfakcja - zdolność oprogramowania do realizacji potrzeb użytkownika w określonym kontekście użycia

Atrybuty, które mogą tutaj być mierzone to:

- zrozumiałość - atrybuty oprogramowania związane z wysiłkiem potrzebnym do poznania logicznej budowy oprogramowania i sposobu jego zastosowania
- łatwość nauki - atrybuty oprogramowania związane z wysiłkiem użytkownika potrzebnym do nauczenia się danej aplikacji
- łatwość obsługi - atrybuty oprogramowania związane z wysiłkiem potrzebnym do efektywnego i wydajnego wykonywania przez użytkownika zadań przy jego pomocy

[8] W wolnym tłumaczeniu: Brytyjski Królewski Narodowy Instytut Niewidomych [przypis SJSI].

- atrakcyjność – zapewnienie użytkownikowi satysfakcji przy z góry określonym użyciu

Ocena użyteczności ma dwa cele:

- usunięcie usterek użyteczności (czasami nazywane oceną kształtującą)
- przetestowanie wymagań użytecznościowych (czasami nazywane oceną podsumowującą)

Testerzy powinni posiadać wiedzę lub doświadczenie w następujących obszarach:

- socjologia
- psychologia
- standardy narodowe (np. Americans Disabilities Act)
- ergonomia

Przeprowadzanie oceny konkretnego wdrożenia powinno być wykonywane w warunkach jak najbardziej zbliżonych do tych, w których system będzie używany. Może to uwzględniać utworzenie laboratorium użyteczności z kamerami, makietami biur, panelami przeglądowymi, użytkownikami itp., tak, aby zespół projektowy mógł obserwować oddziaływanie systemu na prawdziwych ludzi.

Wiele testów użyteczności może być wykonane jako część innych testów, na przykład podczas funkcjonalnych testów systemowych. Wytyczne związane z podejściem do użyteczności mogą być przydatne, aby osiągnąć spójne podejście do wykrywania i raportowania problemów użytecznościowych na każdym etapie cyklu życia.

5.2.5.1 Specyfikacja testów użyteczności

Podstawowe techniki stosowane w testach użyteczności to:

- inspekcja, ocena lub przegląd
- weryfikacja i walidacja rozwiązania końcowego
- przeprowadzenie ankiet i kwestionariuszy

Inspekcja, ocena lub przegląd

Tanim sposobem na wczesne znalezienie problemów może być inspekcja lub przegląd specyfikacji i projektów pod kątem użyteczności przy zaangażowaniu użytkowników.

Do odszukania w projekcie problemów z użytecznością, których rozwiązywanie może stanowić część iteracyjnego procesu projektowania, można zastosować ocenę heurystyczną (systematyczną inspekcję projektu interfejsu użytkownika pod kątem użyteczności). Obejmuje ona ocenę interfejsu i jego zgodności z ustanowionymi zasadami użyteczności (heurystykami), dokonaną przez mały zespół przeglądających.

Walidacja wdrożenia

Do przeprowadzenia walidacji rozwiązania końcowego mogą zostać użyte funkcjonalne testy systemowe, które można przekształcić w scenariusze testów użyteczności. W takich scenariuszach

powinno się skupić na pomiarach konkretnych atrybutów użyteczności, takich jak szybkość uczenia lub łatwość obsługi, a nie na wynikach oczekiwanych przy testach funkcjonalnych.

Scenariusze testowe użyteczności mogą być nakierowane na testowanie składni lub semantyki.

- Składnia: struktura lub gramatyka interfejsu (np. co może zostać wprowadzone w pole wejściowe).
- Semantyka: znaczenie i cel (np. sensowne i niepozbawione znaczenia komunikaty systemowe i wyniki dostarczane użytkownikowi).

Technikami używanymi w projektowaniu tych scenariuszy mogą być:
- metody czarnoskrzynkowe opisane, na przykład, w standardzie BS 7925-2
- przypadki użycia, spisane lub przedstawione w języku UML

Scenariusze testowe w testach użyteczności zawierają instrukcje dla użytkowników, czas na wywiady przed i po testach w celu udzielenia instrukcji i zebrania informacji zwrotnej oraz uzgodniony protokół prowadzenia sesji. Protokół ten powinien zawierać opis tego, jak będą wykonywane testy, przewidywane czasy, robienie notatek i dokumentowanie przebiegu sesji oraz metody wywiadów i badań.

Ankiety i kwestionariusze

Do zebrania obserwacji dotyczących interakcji użytkowników z systemem w laboratorium użyteczności można zastosować techniki związane z kwestionariuszami oraz ankietami.

Ustandaryzowane i ogólnodostępne badania, takie jak SUMI (*Software Usability Measurment Inventory*) lub WAMMI (*Website Analysis and MeasurMent Inventory*) pozwalają na porównanie wyników z bazą danych wcześniej wykonanych pomiarów użyteczności. Dodatkowo SUMI, zapewniające konkretne pomiary użyteczności, może być wykorzystane do określenia kryteriów zakończenia lub akceptacji testów.

5.2.6 Testowanie dostępności (ang. *accessibility testing*)

Bardzo ważne jest uwzględnienie dostępności oprogramowania dla ludzi o szczególnych wymaganiach lub ograniczeniach w jego użyciu. Dotyczy to szczególnie osób niepełnosprawnych. Należy pamiętać o tym, aby brać pod uwagę odpowiednie standardy, takie jak *Web Content Accessibility Guidelines* i akty prawne, takie jak *Disability Discrimination Act* (UK, Australia) i *Section 508* (US).

5.3 Atrybuty jakościowe do testowania technicznego

Atrubuty jakościowe, którymi zajmuje się techniczny analityk testów, skupiają się bardziej na tym, „jak" działa produkt, niż na funkcjonalnych aspektach tego, „co" produkt robi. Te testy mogą być wykonywane na każdym z poziomów testowania, ale są szczególnie istotne dla:

Testowania modułów (istotne dla systemów czasu rzeczywistego i systemów wbudowanych)

- porównywania wydajności
- wykorzystania zasobów

Testów systemowych i operacyjnych testów akceptacyjnych

- obejmują testy atrybutów i podatrybutów jakościowych wymienionych poniżej, zależnie od ryzyka i dostępnych zasobów
- testy techniczne, na tym poziomie, nakierowane są na testowanie konkretnego systemu, na przykład kombinacji sprzętu i oprogramowania, łącznie z serwerami, oprogramowaniem klienckim, bazami danych, sieciami i innymi zasobami

Testy te są często kontynuowane nawet po wdrożeniu oprogramowania na produkcję, zazwyczaj przez osobny zespół lub organizację. Pomiary atrybutów jakościowych zebrane w testach przedwdrożeniowych mogą stanowić bazę dla *SLA* (*Service Level Agreement*) pomiędzy dostawcą i operatorem systemu.

Wyróżniamy następujące atrybuty jakościowe:

- zabezpieczenie techniczne
- niezawodność
- efektywność
- łatwość utrzymania (utrzymywalność, pielęgnowalność)
- przenaszalność

5.3.1 Techniczne testowanie zabezpieczeń (ang. *technical security testing*)

Testowanie zabezpieczeń różni się od innych form testowania dziedzinowego i technicznego w dwóch istotnych aspektach:

1. Zastosowanie standardowych technik wyboru testowych danych wejściowych może spowodować, że nie wykryjemy ważnych braków w zabezpieczeniach.

2. Objawy usterek w zabezpieczeniach różnią się znacznie od tych znajdowanych w innych typach testów.

W zabezpieczeniach oprogramowania istnieje wiele słabych punktów w miejscach, gdzie oprogramowanie jednocześnie funkcjonuje zgodnie z projektem, ale przy okazji wykonuje dodatkowe, niezamierzone działania.

Te efekty uboczne stanowią największe zagrożenie dla bezpieczeństwa oprogramowania. Na przykład odtwarzacz multimediów, który poprawnie odtwarza pliki audio, ale robi to przez zapisanie plików do niezaszyfrowanego tymczasowego katalogu, posiada efekt uboczny, który może być wykorzystany przez *hackerów* oprogramowania.

Głównym zagadnieniem dotyczącym zabezpieczeń jest to, jak ustrzec informację przed niezamierzonym użyciem przez nieuprawnione osoby. Testowanie zabezpieczeń próbuje złamać politykę zabezpieczeń systemu przez ocenę podatności systemu na zagrożenia takie jak:

- nieuprawnione kopiowanie aplikacji lub danych (np. tzw. „piractwo")
- nieuprawniony dostęp (np. możliwość wykonywania zadań, do których użytkownik nie ma uprawnień)
- przepełnienie bufora, które może być spowodowane przez wprowadzenie bardzo długich łańcuchów znaków w pole wejściowe; co z kolei może umożliwić wykonanie „wrogiego" kodu
- odmowa wykonania usługi, która uniemożliwia używanie aplikacji przez użytkowników (np. przez przeciążenie serwera WWW ogromną ilością zapytań, tzw. „nękanie")
- podsłuchiwanie transferów danych przez sieci w celu przechwycenia poufnych danych (np. na temat transakcji kartami płatniczymi)
- łamanie kodów kryptograficznych używanych do ochrony poufnych danych
- bomby logiczne (czasami w USA nazywane „Jajkami Wielkanocnymi"), które mogą być wprowadzane we wrogich zamiarach w kod i które uaktywniają się w pewnych warunkach (np. określonego dnia); bomby logiczne, gdy się uaktywnią, mogą dokonywać szkodliwych działań, np. kasować pliki lub formatować dyski

Poszczególne zagadnienia dotyczące zabezpieczeń mogą być podzielone w następujący sposób:

- związane z interfejsem użytkownika
  - nieautoryzowany dostęp
  - szkodliwe dane wejściowe
- związane z systemem plików
  - dostęp do poufnych danych przechowywanych w plikach i repozytoriach
- związane z systemem operacyjnym
  - przechowywanie poufnych danych, takich jak np. hasła, w niezaszyfrowanej formie w pamięci; złamanie systemu przez wrogie dane wejściowe może spowodować ujawnienie tych informacji
- związane z oprogramowaniem zewnętrznym
  - interakcje, które mogą zajść pomiędzy zewnętrznymi modułami, z których system korzysta; interakcje mogą wystąpić na poziomie sieciowym (np. niepoprawnie przekazywane pakiety lub komunikaty) lub modułów oprogramowania (np. awaria w module, z którego korzysta system).

Trzeba zauważyć, że usprawnienia wykonane w celu poprawy zabezpieczeń systemu mogą wpłynąć na jego wydajność. Po wprowadzeniu takich usprawnień, należy rozważyć powtórzenie testów wydajnościowych.

5.3.1.1 Specyfikacja testów zabezpieczeń

Przy specyfikowaniu testów zabezpieczeń może zostać zastosowane następujące podejście:

- Zdobądź informacje

  Używając szeroko dostępnych narzędzi, sporządź profil lub mapę sieciową systemu. Te informacje mogą zawierać imiona i nazwiska pracowników, adresy fizyczne, szczegóły

wewnętrznych sieci, numery IP, informacje na temat używanego sprzętu i oprogramowania, wersję systemu operacyjnego.

- Znajdź słabe punkty systemu

  Używając szeroko dostępnych narzędzi, przeskanuj system szukając słabych zabezpieczeń. Takich narzędzi nie używa się bezpośrednio do łamania systemów, ale do wyszukiwania słabych punktów, które mogą pozwolić na stworzenie naruszeń w polityce bezpieczeństwa. Niektóre naruszenia mogą zostać wykryte przy użyciu list kontrolnych, które można znaleźć np. pod adresem www.testingstandards.co.uk.

- Opracuj plany ataku

  Opracuj plany ataku, czyli plany działań testowych mających na celu złamanie polityki bezpieczeństwa konkretnego systemu, przy użyciu zdobytych informacji. Żeby wykryć najpoważniejsze usterki w zabezpieczeniach, w planach ataku powinno zostać opracowanych kilka zestawów danych wejściowych dla różnych interfejsów (np. interfejsu użytkownika, systemu plików).

- Wartościowym źródłem pomysłów na techniki testów zabezpieczeń są różne sposoby ataku opisane w [Whittaker04]. Więcej informacji na temat ataków na zabezpieczenia znajduje się w sekcji 4.4.

5.3.2 Testowanie niezawodności (ang. *reliability testing*)

Celem testowania niezawodności jest monitorowanie statystycznych miar dojrzałości oprogramowania w czasie i porównywanie ich z przyjętymi celami niezawodności. Używane miary mogą przybierać formę średniego czasu pomiędzy awariami (ang. *Mean Time Between Failures* - MTBF), średniego czasu naprawy (ang. *Mean Time To Repair* - MTTR) lub dowolną inną miarę częstotliwości występowania awarii (np. liczba awarii określonej wagi na tydzień). Poprawa monitorowanych wartości w czasie może zostać wyrażona w formie modelu wzrostu niezawodności (ang. *Reliability Growth Model*).

Testowanie niezawodności może przybrać formę powtarzanego zbioru wybranych testów, przypadkowych testów wybieranych z określonego zestawu lub testów generowanych na podstawie modelu statystycznego. W rezultacie testy te mogą zająć dużo czasu (liczonego w dniach).

Jako kryterium wyjścia (np. dla wydania produkcyjnego) może zostać użyta analiza miar dojrzałości oprogramowania w czasie. Miary takie, jak *MTBF* i *MTTR* mogą być elementem *SLA* monitorowanym w środowisku produkcyjnym.

Inżynieria i Testowanie Niezawodności Oprogramowania (ang. *Software Reliability Engineering and Testing* - SRET) jest standardowym podejściem do testów niezawodności.

5.3.2.1 Testowanie odporności (ang. *tests for robustness*)

Podczas gdy testy funkcjonalne mogą ocenić tolerancję oprogramowania na błędy w zakresie reakcji na nieprzewidziane wartości wejściowe (tak zwane testy negatywne), testy techniczne oceniają tolerancję systemu na awarie, które zachodzą na zewnątrz testowanego systemu. Takie błędy są zwykle zgłaszane przez system operacyjny (np. brak miejsca na dysku, niedostępny proces lub usługa, nie znaleziono pliku, brak pamięci). Testy tolerancji na usterki na poziomie systemowym mogą być wspierane przez narzędzia.

5.3.2.2 Testy odtwarzalności (ang. *recoverability testing*)

Dalsze formy testowania niezawodności oceniają zdolność systemu do odzyskania możliwości działania po awariach sprzętu lub oprogramowania w sposób, który pozwala na ponowne podjęcie pracy przez system. Testy odtwarzalności zawierają testy poprawnej pracy pomimo usterek (ang. *failover tests*) oraz testy tworzenia kopii zapasowych i ich odtwarzania (ang. *backup & restore tests*).

Testy poprawnej pracy pomimo usterek są przeprowadzane tam, gdzie konsekwencje awarii są tak poważne, że wdrożono mechanizmy programowe lub sprzętowe, aby zapewnić ciągłość pracy systemu nawet w przypadku awarii. Tego rodzaju testy mogą być stosowane na przykład w przypadkach wysokiego ryzyka poniesienia strat finansowych lub w sytuacjach, kiedy awaria systemu może narażać kogoś na niebezpieczeństwo. Tam, gdzie awarie mogą być spowodowane katastrofami, ta forma testów odtwarzalności nazywana jest testowaniem odtwarzania po katastrofie (ang. *disaster recovery*).

Typowymi miarami sprzętowymi stosowanymi tutaj mogą być: rozłożenie obciążenia pomiędzy kilkoma procesorami, klastrami, procesorami lub dyskami, które natychmiast przejmują działanie, kiedy jeden z nich się zepsuje (np. *Redundant Array of Inexpensive Disks - RAID*). Typowymi miarami programowymi mogą być: implementacja kilku niezależnych instancji systemu (na przykład w systemach kontroli lotu) w tak zwanych niepodobnych systemach nadmiarowych (ang. *redundant dissimiliar systems*). Systemy nadmiarowe stanowią zwykle kombinację elementów programowych i sprzętowych; wyróżniamy systemy podwójne, potrójne lub poczwórne w zależności od ilości niezależnych instancji (odpowiednio 2, 3 lub 4). Aspekt braku podobieństwa jest osiągany przez dostarczenie tych samych wymagań do dwóch (lub więcej) niezależnych i niepołączonych ze sobą zespołów projektowych w celu zbudowania różnego oprogramowania dostarczającego tych samych usług. Chroni to niepodobne systemy nadmiarowe zmniejszając prawdopodobieństwo wywołania tego samego efektu w przypadku niepoprawnych danych wejściowych. Kroki podjęte w celu zwiększenia odtwarzalności systemu mogą mieć bezpośredni wpływ na jego niezawodność, a co za tym idzie mogą być brane pod uwagę podczas wykonywania testów niezawodności.

Testowanie poprawnej pracy pomimo usterek zostało zaprojektowane do testowania systemów przez symulację różnych rodzajów awarii lub wykonywanie ich w kontrolowanym środowisku.

Aby upewnić się, że dane nie zostały utracone lub zniszczone oraz że zostały utrzymane uzgodnione poziomy świadczenia usług (np. dostępność funkcji, czasy odpowiedzi), zaraz po wystąpieniu awarii testowane są mechanizmy poprawnej pracy pomimo usterek.

Więcej informacji na temat testów poprawnej pracy pomimo usterek można znaleźć na stronach www.testingstandards.co.uk.

W ramach testów tworzenia i odtwarzania kopii zapasowych skupiamy się na procedurach ustanowionych, aby zminimalizować skutki awarii. Takie testy oceniają procedury (zwykle spisane w podręcznikach) tworzenia różnych form kopii zapasowych i ich odtwarzania w przypadku utraty lub zniszczenia danych. Przypadki testowe są tak zaprojektowane, by zapewnić, że pokryte są wszystkie krytyczne ścieżki procedury. Można przeprowadzać przeglądy techniczne („na sucho") i sprawdzać zgodność podręczniki z właściwą procedurą instalacji. Operacyjne testy akceptacyjne polegają na wykonaniu scenariuszy w środowisku produkcyjnym lub zbliżonym do produkcyjnego w celu walidacji ich rzeczywistego wykorzystania.

Testy tworzenia kopii zapasowych i ich odtwarzania mogą uwzględniać:

- czas potrzebny na wykonanie różnych typów kopii zapasowych (np. pełnych i przyrostowych)
- czas potrzebny na odtworzenie danych
- poziomy gwarantowanych kopii danych (np. odtworzenie wszystkich danych nie starszych, niż 24 godziny, odtworzenie niektórych transakcji nie starszych, niż jedna godzina)

5.3.2.3 Specyfikacja testów niezawodności

Testy niezawodności w większości opierają się na wzorcach użycia (czasami nazywanych profilami operacyjnymi) i mogą być przeprowadzane formalnie lub zależnie od ryzyka. Dane testowe mogą być generowane przy użyciu metod przypadkowych lub pseudoprzypadkowych.

Wybór konkretnej krzywej wzrostu niezawodności powinien być uzasadniony. Do przeanalizowania danych na temat awarii i ustalenia, która z krzywych wzrostu niezawodności najlepiej pasuje do aktualnie dostępnych danych można użyć narzędzi.

W ramach testów niezawodności można szukać w szczególności wycieków pamięci. Specyfikacja takich testów zakłada wielokrotne wykonywanie tych samych operacji intensywnie wykorzystujących pamięć , aby upewnić się, że alokowana pamięć jest poprawnie zwalniana.

5.3.3 Testowanie efektywności (ang. *efficiency testing*)

Efektywność jako atrybut jakości jest oceniana przez przeprowadzenie testów nakierowanych na zachowanie systemu związane z czasem i zasobami. Testowanie efektywności w aspekcie czasowym zostało omówione poniżej jako testowanie wydajnościowe, testowanie obciążeniowe, testowanie przeciążające oraz testowanie skalowalności.

5.3.3.1 Testowanie wydajnościowe (ang. *performance testing*)

Ogólne pojęcie testowania wydajnościowego może zostać podzielone na kilka różnych typów testów w zależności od wymagań niefunkcjonalnych, które są testowane. Te typy testów obejmują: wydajność, obciążenie, przeciążenie oraz skalowalność.

Testowanie wydajnościowe skupia się na zdolności modułu lub systemu do wygenerowania odpowiedzi na dane wejściowe od użytkownika lub innego systemu, w określonym czasie, w określonych warunkach (patrz też: testy obciążeniowe i testy przeciążające poniżej). Pomiary wydajności różnią się w zależności od celu testu. Dla pojedynczych modułów programowych wydajność może być mierzona w cyklach procesora, a dla systemów z klientami - wydajność może być mierzona jako czas odpowiedzi na konkretne żądanie użytkownika. Dla systemów składających się z kilku modułów (np. klientów, serwerów, baz danych) pomiary wydajności są wykonywane pomiędzy poszczególnymi modułami, tak żeby można było wykryć „wąskie gardła" w przetwarzaniu.

5.3.3.2 Testowanie obciążeniowe (ang. *load testing*)

Testowanie obciążeniowe skupia się na testowaniu zdolności systemu do obsługi coraz większego, przewidywanego poziomu obciążenia, wynikającego z transakcji wygenerowanych przez wielu równolegle pracujących użytkowników. Można tutaj mierzyć i analizować średni czas odpowiedzi dla użytkowników wykonujących różne typowe scenariusze użycia (profile operacyjne). Zobacz również [Splaine01].

Istnieją dwa podtypy testów obciążeniowych: testowanie przez wielu użytkowników (ang. *multi-user testing)* z realistyczną liczbą użytkowników oraz testowanie obciążenia (ang. *volume testing*) z bardzo dużą liczbą użytkowników. Podczas testów obciążeniowych sprawdza się zarówno czas odpowiedzi na żądanie, jak i obciążenie sieci.

5.3.3.3 Testowanie przeciążające (ang. *stress testing*)

Testowanie przeciążające sprawdza zdolność systemu do obsługi szczytowych wartości obciążenia, na granicy lub poza granicą maksymalnej wytrzymałości. W miarę podnoszenia obciążenia wydajność systemu powinna się obniżać powoli i w sposób przewidywalny, bez wystąpienia awarii. Obciążony system powinien zostać przetestowany w szczególności pod względem integralności funkcjonalnej w celu znalezienia możliwych usterek w przetwarzaniu oraz niespójności w danych.

Jednym z możliwych celów testowania przeciążającego jest znalezienie granicy, poza którą system przestaje działać; dzięki czemu można określić "najsłabsze ogniwo łańcucha". Testowanie przeciążające pozwala na dodawanie na czas dodatkowych komponentów do systemu (np. pamięci, mocy procesora, miejsca w bazie danych).

W „testowaniu szczytowym" (ang. *spike testing*) symulowane są kombinacje warunków, które mogą spowodować nagłe, znaczne zwiększenie obciążenia systemu. „Testy podskokowe" (ang. *bounce tests)* aplikują systemowi kilka takich szczytów z okresami niskiego obciążenia pomiędzy nimi. Takie testy pozwalają na sprawdzenie, jak zachowa się system podczas zmian obciążenia i czy jest w stanie pobierać i zwalniać zasoby w miarę potrzeb. Patrz też [Splaine01].

5.3.3.4 Testy skalowalności (ang. *scalability testing*)

Testowanie skalowalności skupia się na sprawdzeniu zdolności systemu do realizacji przyszłych wymagań na efektywność, które to wymagania mogą przekraczać obecne założenia. Celem tych

testów jest ocena zdolności systemu do rozrostu (np. zwiększenie liczby użytkowników, bądź ilości składowanych danych), bez przekraczania uzgodnionych ograniczeń i bez awarii. Gdy ograniczenia są już znane, można ustawić i monitorować wartości progowe na systemie produkcyjnym, w celu wygenerowania ostrzeżenia o nadchodzących problemach.

5.3.3.5 Test wykorzystania zasobów (ang. *test of resource utilization*)

Jest to test efektywności sprawdzający użycie zasobów systemowych (np. pamięci, dysku i sieci). Porównaniu podlega zużycie zasobów systemowych przy typowym obciążeniu oraz obciążeniu dodatkowym (np. duża ilość transakcji lub danych).

Na przykład dla systemów wbudowanych czasu rzeczywistego w testach wydajnościowych dużą rolę odgrywa poziom zużyci pamięci (czasami nazywane ang. *memory footprint*).

5.3.3.6 Specyfikacja testów efektywności

Specyfikacja testów dla różnych typów testów efektywności, takich jak testy wydajności, obciążenia oraz przeciążające bazuje na zdefiniowaniu profili operacyjnych. Reprezentują one różne formy zachowania użytkownika podczas interakcji z aplikacją. Dla jednej aplikacji może wystąpić kilka profili operacyjnych.

Liczba użytkowników przypadających na jeden profil operacyjny może zostać określona przy pomocy narzędzi do monitorowania (gdy dostępna jest już docelowa lub porównywalna aplikacja) lub przez przewidywanie użycia. Przewidywania dokonuje się na bazie algorytmów lub na podstawie informacji dostarczonej przez ”biznes”. Przewidziane wartości są niezwykle istotne w specyfikowaniu profili operacyjnych dla testów skalowalności.

Profile operacyjne stanowią podstawę dla przypadków testowych i zwykle generuje się je przy użyciu narzędzi testowych. W takich przypadkach używa się zwykle pojęcia "wirtualny użytkownik", które oznacza użytkownika symulowanego w ramach profilu operacyjnego.

5.3.4 Testowanie pielęgnowalności (ang. *maintainability testing*)

Testowanie pielęgnowalności odnosi się w ogólności do łatwości analizowania, zmiany i testowania oprogramowania. Prawidłowe techniki testowania pielęgnowalności obejmują analizę statyczną i listy kontrolne.

5.3.4.1 Dynamiczne testowanie pielęgnowalności

Dynamiczne testowanie pielęgnowalności skupia się na procedurach spisanych w celu pielęgnacji konkretnej aplikacji (np. wykonywania aktualizacji oprogramowania). Wybrane scenariusze pielęgnacyjne są wykorzystywane jako przypadki testowe w celu udowodnienia, że uzgodnione poziomy świadczenia usług są utrzymywane przy zastosowaniu przewidzianych procedur.

Taka forma testowania ma szczególne znaczenie, gdy infrastruktura jest złożona, zaś procedury wsparcia angażują wiele działów lub organizacji. Może ona mieć miejsce jako część akceptacyjnych testów operacyjnych. [www.testingstandards.co.uk]

5.3.4.2 Analizowalność (pielęgnacja korekcyjna)

Ta forma testów pielęgnowalności polega na mierzeniu czasu potrzebnego na zdiagnozowanie i poprawienie usterek występujących w systemie. W tym przypadku prostą miarę może stanowić średni czas zdiagnozowania i poprawienia zidentyfikowanej usterki.

5.3.4.3 Zmienialność, stabilność i testowalność (pielęgnacja adaptacyjna)

Pielęgnowalność systemu może być również mierzona jako pracochłonność wymagana do wykonania zmian w systemie (np. zmian w kodzie). Ponieważ pracochłonność zależy od wielu czynników, takich jak metodyka projektowania (np. programowanie obiektowe), standardy kodowania itp., ta forma testów pielęgnowalności może być wykonana jako analiza lub przegląd. Testowalność odnosi się konkretnie do pracochłonności testowania wprowadzonych zmian. Stabilność - do tego, jak system reaguje na wprowadzone zmiany. W systemach o niskiej stabilności występuje wiele problemów wtórnych (zwanych również efektem domino) po wykonaniu zmian. [ISO 9126] [www.testingstandards.co.uk]

5.3.5 Testowanie przenaszalności (ang. *portability testing*)

Testowanie przenaszalności w ogólności odnosi się do łatwości transferu oprogramowania na środowisko przeznaczenia, czy to pierwotnie, czy z istniejącego środowiska. Na testy przenaszalności składają się testy instalowalności, koegzystencji (współpracy), zdolności adaptacyjnej oraz zastępowalności.

5.3.5.1 Testowanie instalowalności (ang. *installability testing*)

Testowanie instalowalności przeprowadza sie na oprogramowaniu używanym do zainstalowania innego oprogramowania na środowisku docelowym. Może to na przykład obejmować oprogramowanie instalujące system operacyjny na procesorze lub kreator instalujący produkt na komputerze klienta. Typowymi celami testowania instalowalności są:

- sprawdzenie czy oprogramowanie może zostać zainstalowane przy pomocy instrukcji zawartych w podręczniku instalacji (włączając w to uruchomienie skryptów instalacyjnych) lub przy użyciu instalatora. Powyższe obejmuje również sprawdzenie opcji instalacji dla różnych konfiguracji programowo-sprzętowych i dla różnych stopni instalacji (np. pełnej lub częściowej)
- testowanie, czy awarie podczas instalacji (np. brak możliwości załadowania jakiejś biblioteki) są traktowane przez instalator poprawnie, bez pozostawiania systemu w niezdefiniowanym stanie (np. częściowo zainstalowanego albo niepoprawnie skonfigurowanego)
- testowanie, czy można przeprowadzić częściową instalację lub deinstalację
- testowanie, czy instalator jest w stanie rozpoznać nieprawidłowe platformy sprzętowe lub konfiguracje systemów operacyjnych

- pomiar tego, czy proces instalacji może zostać zakończony w ciągu przewidzianego czasu lub założonej ilości kroków
- sprawdzenie, czy oprogramowanie może zostać cofnięte do starszej wersji lub czy też można je w ogóle odinstalować

Po instalacji wykonuje się zwykle testy funkcjonalności, aby wykryć ewentualne usterki, które mogły zostać wprowadzone podczas instalacji (np. niepoprawna konfiguracja, niedostępne funkcje). Równolegle z testowaniem instalowalności wykonuje się testy użyteczności, by sprawdzić, czy instrukcje oraz odpowiedzi instalatora i komunikaty o błędach podczas instalacji są zrozumiałe dla użytkowników.

5.3.5.2 Testowanie koegzystencji (ang. *co-existence*)

Systemy, które nie są związane ze sobą, nazywane są zgodnymi, jeżeli mogą działać w tym samym środowisku (np. na tym samym sprzęcie) bez wzajemnego wpływania na swoje zachowanie (np. bez konfliktu o zasoby). Testy współpracy można przeprowadzić na przykład, gdy nowy lub zaktualizowany system jest wdrażany na środowiska, na których już zostały zainstalowane inne aplikacje.

Problemy we współpracy mogą się pojawić, gdy aplikacja jest testowana na środowisku, w którym jest jedyną zainstalowaną aplikacją (gdzie tych problemów nie da się wykryć), a później zostaje zainstalowana na innym środowisku (np. na środowisku produkcyjnym), na którym działają też inne aplikacje.

Celami testów współpracy mogą być:

- ocena możliwego negatywnego wpływu na funkcjonalność, gdy aplikacje są załadowane na to samo środowisko (np. konflikt użycia zasobów, gdy na serwerze uruchomionych jest wiele aplikacji)
- ocena wpływu instalacji poprawek i uaktualnień systemu operacyjnego na zainstalowane aplikacje

Testowanie współpracy jest wykonywane zwykle po pozytywnym zakończeniu testów systemowych i testów akceptacyjnych przez użytkownika.

5.3.5.3 Testowanie zdolności adaptacyjnej (ang. *adaptability testing*)

Testowanie zdolności do adaptacji sprawdza, czy dana aplikacja może funkcjonować poprawnie we wszystkich założonych środowiskach docelowych (sprzęt, oprogramowanie, warstwa integracyjna, system operacyjny itd.). Projektowanie testów sprawdzających zdolność adaptacyjną systemu wymaga zidentyfikowania kombinacji wyżej wymienionych czynników składających się na środowiska docelowe, skonfigurowania tych środowisk oraz ich udostępnienia zespołowi testowemu. Środowiska te są później testowane przy użyciu podzbioru testów funkcjonalnych, które wykorzystują różne moduły obecne w środowisku.

Zdolność adaptacyjna może odnosić się do możliwości przeniesienia oprogramowania na różne określone środowiska przez wykonanie zdefiniowanej procedury. Testy mogą oceniać tę procedurę.

Testy zdolności adaptacyjnej mogą być wykonywane łącznie z testami instalowalności. Po tych testach zwykle wykonuje się testy funkcjonalne, aby wykryć usterki, które mogły zostać wprowadzone przy adaptowaniu oprogramowania do innego środowiska.

5.3.5.4 Testowanie zastępowalności (ang. *replaceability testing*)

Zastępowalność polega na możliwości wymiany modułów programowych w systemie na inne. Powyższe może mieć szczególnie istotne znaczenie dla systemów, które jako modułów używają oprogramowania „z półki” (COTS).

Testy zastępowalności mogą być wykonywane równolegle z funkcjonalnymi testami integracji podstawowych funkcjonalności systemu, gdy więcej, niż jeden wymienny moduł jest dostępny do integracji. Zastępowalność może być oceniania w ramach przeglądu technicznego lub inspekcji, które kładą szczególny nacisk na jasną definicję interfejsów z potencjalnymi modułami wymienialnymi.

6. Przeglądy

Terminologia

audyt, IEEE 1028, przegląd nieformalny, inspekcja, prowadzący inspekcję, przegląd kierowniczy, moderator, przegląd, przeglądający, przegląd techniczny, przejrzenie

6.1 Wprowadzenie

Skuteczny proces przeglądu wymaga planowania, uczestnictwa i śledzenia wyników. Dostawcy szkoleń muszą upewnić się, że kierownicy testów rozumieją swoją odpowiedzialność w zakresie czynności planowania przeglądu i śledzenia jego wyników. Testerzy muszą być aktywnymi uczestnikami przeglądów, prezentując swój subiektywny punkt widzenia. Testerzy powinni być przeszkoleni, żeby lepiej rozumieć swoją rolę w przeglądach technicznych. Wszystkim uczestnikom przeglądu musi zależeć na korzyściach płynących z dobrze przeprowadzonego przeglądu technicznego. Poprawnie wykonane inspekcje najlepiej i w najbardziej opłacalny sposób przyczyniają się do zapewnienia jakości dostarczanego produktu. Międzynarodowym standardem dotyczącym przeglądów jest IEEE 1028.

6.2 Zasady przeprowadzania przeglądów

Przegląd jest jednym ze sposobów testowania statycznego. Często głównym celem przeglądów jest wykrywanie defektów. Przeglądający znajdują defekty przez bezpośrednie sprawdzanie dokumentów.

Podstawowe typy przeglądów są opisane w rozdziale 3.2 syllabusa poziomu podstawowego (w wersji 2005) i są powtórzone tutaj, w podrozdziale 6.3.

Wszystkie rodzaje przeglądów należy wykonywać, gdy tylko dostępne są dokumenty źródłowe (dokumenty opisujące wymagania projektowe) oraz standardy, które musi spełniać projekt. Jeżeli brakuje jednego z dokumentów, bądź standardów, wtedy można wykryć usterki i niespójności tylko w pojedynczych dokumentach, a nie w całości dokumentacji. Dokument do przejrzenia musi zostać dostarczony przeglądającym odpowiednio wcześnie, tak aby zdążyli zapoznać się z jego treścią.

Przedmiotem przeglądu mogą być wszystkie typy dokumentów np. kod źródłowy, specyfikacja wymagań, pomysły, plany testów, dokumenty testowe. Po przeglądzie kodu źródłowego zwykle wykonuje się testy dynamiczne, które są tak zaprojektowane, żeby wykrywały błędy, których nie da się znaleźć za pomocą testów statycznych.

Przegląd może mieć jeden z trzech wyników:

- dokument może być używany bez zmian lub z małymi zmianami
- dokument musi zostać zmieniony, ale dodatkowy przegląd nie jest wymagany
- dokument musi zostać zmieniony w znacznym stopniu, a potwórny przegląd jest konieczny

Role i odpowiedzialność osób zaangażowanych w typowy przegląd są opisane w syllabusie poziomu podstawowego, np. kierownik, moderator lub prowadzący, autor, przeglądający oraz protokolant. Inni, którzy mogą brać udział w przeglądach to osoby decyzyjne, interesariusze oraz przedstawiciele klienta lub użytkowników. Dodatkową, opcjonalną rolą pojawiającą się czasami w inspekcjach może być referent, który ma parafrazować fragmenty przeglądanego produktu podczas spotkania. W dodatku do ról przeglądowych niektórzy z przeglądających mogą mieć przyznane role związane z wyszukiwaniem usterek konkretnego typu.

Jeden produkt może podlegać więcej niż jednemu typowi przeglądu. Na przykład zespół może przeprowadzić przegląd techniczny, żeby ustalić które funkcje mają zostać zaimplementowane w następnej iteracji, a następnie specyfikacja wybranych funkcjonalności może podlegać inspekcji.

6.3 Rodzaje przeglądów

W syllabusie poziomu podstawowego wprowadzono następujące rodzaje przeglądów:

- przegląd nieformalny
- przejrzenie
- przegląd techniczny
- inspekcja

W praktyce można również organizować hybrydy powyższych typów np. przegląd techniczny z zestawem reguł.

6.3.1 Przegląd kierowniczy i audyt

Standard IEEE 1028 opisuje dwa dodatkowe typy przeglądów ponad te omówione w syllabusie poziomu podstawowego:

- przegląd kierowniczy
- audyt

Podstawowymi cechami przeglądu kierowniczego są:

- jego głównym celem jest monitorowanie postępu, ocena statusu i podejmowanie decyzji o przyszłych działaniach
- jest przeprowadzany przez lub dla kierownictwa bezpośrednio odpowiedzialnego za projekt lub system
- jest przeprowadzany przez lub dla interesariusza lub osoby decyzyjnej np. kierownika wyższego szczebla lub dyrektora
- sprawdza zgodność z planami lub adekwatność procedur zarządczych
- zawiera ocenę ryzyka projektowego
- jego wynik zawiera zestawienie rzeczy do zrobienia oraz problemów do rozwiązania
- jego uczestnicy powinni być przygotowani, a decyzje są udokumentowane

Należy zauważyć, że kierownicy testów powinni uczestniczyć w przeglądach kierowniczych dotyczących postępu testowania, mogą je również zaproponować.

Audyty są bardzo formalne i zwykle są przeprowadzane, aby zademonstrować zgodność z jakimś zbiorem wymagań, najczęściej z jakimś standardem lub wymaganiami kontraktowymi. Jako takie, audyty są najmniej skuteczne w znajdowaniu usterek.

Główne cechy audytów to:

- głównym ich celem jest dostarczenie niezależnej oceny zgodności z procesem, obowiązującym prawem, standardami itp.
- główny audytor jest odpowiedzialny za audyt i pełni rolę moderatora
- audytorzy zbierają dowody przez przeprowadzanie wywiadów, obserwacje i sprawdzanie dokumentów
- wynikiem audytu są obserwacje, rekomendacje, działania naprawcze oraz podsumowanie (wynik pozytywny lub negatywny)

6.3.2 Przeglądy poszczególnych produktów projektu

Przeglądy mogą być opisywane w kontekście produktów lub działań, które im podlegają. Mogą to być:

- przegląd wymagań kontraktowych
- przegląd wymagań
- przegląd projektu
  - przegląd projektu wysokopoziomowego
  - przegląd projektu szczegółowego
- przegląd akceptacyjny
- przegląd gotowości operacyjnej

Przegląd wymagań kontraktowych może być powiązany z kamieniem milowym przygotowywania kontraktu i jest zwykle przeglądem kierowniczym dla systemów krytycznych ze względu na bezpieczeństwo lub systemów, dla których bezpieczeństwo jest istotne. Biorą w nim udział kierownicy, klienci oraz personel techniczny.

Przegląd wymagań może zostać przeprowadzony w formie przejrzenia, przeglądu technicznego lub inspekcji i może brać pod uwagę wymagania dotyczące bezpieczeństwa i niezawodności, jak również wymagania funkcjonalne i niefunkcjonalne. Przegląd wymagań może wykorzystywać kryteria akceptacji i warunki testowe.

Przeglądy projektu są zwykle przeglądami technicznymi lub inspekcjami i angażują personel techniczny oraz klientów lub interesariuszy. W ramach przeglądu projektu wysokopoziomowego proponuje się pierwsze podejście do niektórych aspektów projektu oraz testów. Przegląd projektu szczegółowego dotyczy wszystkich zaproponowanych rozwiązań projektowych, włączając w to przypadki testowe i procedury.

Przeglądy akceptacyjne mają na celu akceptację systemu przez kierownictwo. Są czasami nazywane przeglądami kwalifikacyjnymi i zwykle przybierają formę przeglądów kierowniczych lub audytów.

6.3.3 Przeprowadzanie formalnych przeglądów

Syllabus poziomu podstawowego przedstawia sześć faz przeglądu formalnego: planowanie, rozpoczęcie, przygotowanie indywidualne, spotkanie przeglądowe, obróbka i sprawdzenie. Zarówno rodzaj przeglądu, jak i przeglądający, powinni być odpowiednio dobrani do przeglądanego produktu, np. plan testów powinien być przeglądany przez kierownika testów, wymagania biznesowe lub projekt testów przez analityka testów, specyfikacja funkcjonalna, przypadki testowe lub skrypty testowe przez technicznego analityka testów.

6.4 Wdrażanie przeglądów

Chcąc skutecznie wdrożyć przeglądy w organizacji, należy wykonać następujące kroki (niekoniecznie w poniższej kolejności):

- zdobyć wsparcie kierownictwa
- przekazać kierownikom informacje o kosztach, korzyściach i problemach we wdrażaniu przeglądów
- wybrać i spisać procedury prowadzenia przeglądów, formularzy i infrastruktury (np. bazy danych pomiarów przeglądów)
- przeprowadzić szkolenie z technik i procedur przeglądów
- uzyskać wsparcie tych, którzy będą wykonywali przeglądy i tych, których produkty będą podlegały przeglądom
- przeprowadzić pilotażowe przeglądy
- zademonstrować oszczędności wynikających z przeglądów
- zastosować przeglądy do najważniejszych dokumentów np. wymagań, umów, planów itd.

Do oceny skuteczności wdrożonych przeglądów można użyć metryk mówiących o zmniejszeniu lub uniknięciu kosztów naprawy defektów i ich konsekwencji. Oszczędności mogą również być mierzone poprzez pomiar czasu zaoszczędzonego przez wczesne znalezienie i usunięcie defektów.

Procesy prowadzenia przeglądów powinny być ciągle monitorowane i usprawniane. Kierownicy powinni być świadomi tego, że poznanie nowej techniki przeglądu jest inwestycją – korzyści z niej nie są natychmiastowe, ale z czasem znacząco wzrastają.

6.5 Czynniki sukcesu w kontekście przeglądów

Wiele czynników składa się na sukces przeglądów. Prowadzenie przeglądów nie musi być trudne, ale przeglądy mogą nie udawać się pod różnymi względami, jeżeli niżej wymienione czynniki nie zostaną wzięte pod uwagę.

Czynniki techniczne:

- upewnić się, że proces przeglądu jest stosowany poprawnie, szczególnie przy bardziej formalnych typach przeglądów, takich jak inspekcje

- zapisywać koszty przeglądu (takie jak ich pracochłonność) oraz osiągane korzyści
- przeglądać wczesne wersje robocze lub kawałki dokumentów, żeby zidentyfikować wzorce defektów, zanim zostaną wbudowane w cały dokument
- upewnić się, że dokumenty lub kawałki dokumentów są przygotowane do przeglądu, zanim przegląd się rozpocznie (np. stosuj kryteria wejścia)
- używać właściwych dla swojej organizacji list kontrolnych najczęściej spotykanych defektów
- używać wielu typów przeglądów w zależności od celu, jaki chce się osiągnąć np. „wyczyszczenie" dokumentu, poprawki techniczne, przepływ informacji, czy zarządzanie postępem prac
- poddawać przeglądom i inspekcjom dokumenty, od których zależą ważne decyzje, na przykład należy wykonać inspekcję oferty, umowy lub wymagań wysokiego poziomu przed przeglądem kierowniczym akceptującym duże wydatki w projekcie
- sprawdzać ograniczony podzbiór dokumentu przy ocenie ogólnej, ale nie przy „czyszczeniu" dokumentu
- namawiać do znajdowania najważniejszych błędów: skupiać się na treści, a nie na formie
- ciągle ulepszać proces prowadzenia przeglądu

Czynniki organizacyjne:

- upewnić się, że kierownicy dają odpowiedni czas na czynności związane z przeglądami, nawet pod presją terminów
- pamiętać, że czas i koszt przeglądów nie są proporcjonalne do znalezionych defektów
- zapewnić odpowiednią ilość czasu na poprawienie znalezionych defektów
- nigdy nie używać metryk z przeglądów do oceny indywidualnej efektywności ludzi
- zapewnić, żeby właściwi ludzie byli przypisani do różnych rodzajów przeglądów
- organizować szkolenia z przeglądów, a w szczególności z bardziej formalnych ich typów
- wspierać forum prowadzących przeglądy oraz dzielenie się pomysłami i doświadczeniami
- upewnić się, że każdy uczestniczy w przeglądach oraz że dokumenty wszystkich są przeglądane
- zastosować najsilniejsze techniki przeglądów do najważniejszych dokumentów
- zapewnić wyważony zespół przeglądających i to, że jego uczestnicy posiadają różne umiejętności i doświadczenie
- wspierać działania zmierzające do usprawnienia procesu, tak żeby zająć się problemami systemowymi
- pokazywać ulepszenia osiągnięte dzięki przeglądom

Czynniki ludzkie:

- nauczyć interesariuszy, żeby spodziewali się wykrycia defektów i przeznaczyli czas na ich usunięcie i ponowny przegląd
- zagwarantować, aby przegląd był pozytywnym doświadczeniem dla autora
- traktować znajdowanie defektów jako coś pożądanego w atmosferze wolnej od obwiniania
- zapewnić konstruktywny, pomocny i obiektywny ton komentarzy, a unikać subiektywizmu
- nie prowadzać przeglądu, jeżeli autor tego nie chce lub jest niechętny
- zachęcać wszystkich do głębokiego przemyślenia najważniejszych aspektów przeglądanych dokumentów

Więcej informacji o przeglądach można znaleźć w [Gilb93] oraz [Wiegers02].

7. Zarządzanie incydentami

Terminologia

IEEE 829, IEEE 1044, IEEE 1044.1, anomalia, rada kontroli zmian, defekt, błąd, awaria, incydent, rejestracja incydentu, priorytet, analiza przyczyny podstawowej , ważność

7.1 Wprowadzenie

Kierownicy testów i testerzy muszą być zaznajomieni z procesem zarządzania defektami. Kierownicy testów powinni skoncentrować się na procesie, włączając w to metody rozpoznawania, śledzenia i usuwania defektów. Testerzy zajmują się głównie dokładnym rejestrowaniem problemów znalezionych w obszarach, które testują. Na każdym etapie cyklu życia oprogramowania, analitycy testów oraz techniczni analitycy testów będą dążyli do czegoś innego. Analityk testów będzie dokonywał oceny systemu z biznesowego punktu widzenia i mając na względzie wymagania użytkownika, np. czy użytkownik będzie wiedział, co zrobić, gdy otrzyma dany komunikat, bądź gdy aplikacja zachowa się w jakiś określony sposób. Techniczny analityk testów będzie oceniał same zachowanie aplikacji, a w przypadku wystąpienia problemu, dokona jego analizy od strony technicznej, np. poprzez próbę wywołania tej samej awarii na różnych platformach, czy też z różnymi ustawieniami pamięci.

7.2 Kiedy defekt może zostać znaleziony

Incydent jest nieoczekiwanym zdarzeniem, które wymaga dalszego zbadania. Incydent jest wykryciem awarii spowodowanej przez defekt. Incydent może, ale nie musi, skutkować utworzeniem nowego zgłoszenia defektu. Defekt jest to realnie występujący problem, który musi zostać rozwiązany poprzez wykonanie zmian w produkcie.

Defekt może zostać wykryty podczas testów statycznych. Awaria może zostać wykryta tylko podczas testów dynamicznych. Każda z faz cyklu życia oprogramowania powinna dostarczać odpowiednich metod wykrywania i eliminowania potencjalnych awarii. Dla przykładu, podczas fazy implementacji do znajdowania i usuwania defektów powinny zostać użyte przeglądy kodu i projektu rozwiązania. Podczas testów dynamicznych, do znajdowania awarii wykorzystywane są przypadki testowe. Im wcześniej defekt jest wykryty i usunięty, tym niższy jest koszt jakości systemu. Należy pamiętać, że defekty mogą istnieć zarówno w testaliach, jak i w przedmiocie testów.

7.3 Cykl życia defektu

Wszystkie defekty mają cykl życia, aczkolwiek w przypadku niektórych – wybrane etapy cyklu życia można pominąć. Cykl życia defektu (opisany w IEEE 1044-1993) składa się z 4 kroków:

- Krok 1: Rozpoznanie
- Krok 2: Badanie
- Krok 3: Działanie
- Krok 4: Dyspozycja

W każdym z tych kroków należy wykonać 3 poniższe czynności:

- zarejestrować
- zaklasyfikować
- zidentyfikować wpływ

7.3.1 Krok 1: Rozpoznanie

Z rozpoznaniem mamy do czynienia, gdy zostaje odkryty potencjalny defekt (incydent). Ten krok może pojawić się we wszystkich fazach cyklu wytwarzania oprogramowania. W momencie rozpoznania, powinny zostać zapisane wszystkie informacje określające defekt. Do takich informacji zaliczamy: dane o środowisku, w którym defekt został odkryty, dane osoby, która odkryła defekt, opis defektu, czas i dane dostawcy (jeśli ma zastosowanie). Rozpoznanie jest definiowane przez określenie wybranych atrybutów potencjalnego defektu, włączając w to czynność projektową, fazę projektu, prawdopodobną przyczynę, powtarzalność, symptomy oraz status produktu będący wynikiem anomalii. Na podstawie tych informacji następuje ocena wpływu defektu poprzez określenie jej wagi, wpływu na harmonogram oraz koszt projektu.

7.3.2 Krok 2: Badanie

Po rozpoznaniu, każdy potencjalny defekt jest badany. W tym kroku ma miejsce odszukanie powiązanych defektów i zaproponowanie rozwiązań, włączając w to brak jakiegokolwiek działania (np. w sytuacji, gdy potencjalny defekt nie jest już aktualny). W tym kroku zapisywane są również dodatkowe informacje o defekcie, przeprowadzana jest jego ponowna klasyfikacja, jak również ocena wpływu informacji dostarczonych w poprzednim kroku.

7.3.3 Krok 3: Działanie

Krok 3 – Działanie jest przeprowadzany w oparciu o informacje dostarczone w poprzednim kroku. Działanie zawiera w sobie zarówno akcje niezbędne do usunięcia danego defektu, jak i akcje mające na celu przegląd/poprawę istniejących procesów i polityk, tak aby zapobiec pojawieniu się podobnych defektów w przyszłości. Dla każdej zmiany muszą być wykonane testy regresji, retesty oraz testy progresywne. W tym kroku zapisywane są również dodatkowe informacje o defekcie, przeprowadzana jest jego ponowna klasyfikacja, jak również ocena wpływu informacji dostarczonych w poprzednim kroku.

7.3.4 Krok 4: Dyspozycja

Następnie anomalia przechodzi do kroku Dyspozycji, gdzie zapisywane są wszelkie dodatkowe informacje a defekt klasyfikowany jest jako: zamknięty, odroczony, włączony lub odnoszący się do innego projektu.

7.4 Atrybuty defektu

IEEE 1044-1993 określa zestaw obowiązkowych atrybutów, które są wypełniane w różnych momentach cyklu życia defektu. Standard definiuje również długą listę atrybutów opcjonalnych. Zgodnie z IEEE 1044.1, podczas wdrażania standardu IEEE 1044-1993 dopuszczalne jest tworzenie mapowania pomiędzy terminami z IEEE, a tymi używanymi w konkretnej firmie. To pozwala na zachowanie zgodności z IEEE 1044-1993 bez zbyt sztywnego trzymania się przyjętej konwencji nazewniczej. Zachowanie zgodności z IEEE pozwala na dokonanie porównania pomiędzy przechowywanymi informacjami dotyczącymi defektu w wielu firmach i organizacjach.

Niezależnie od tego, czy zgodność z IEEE jest celem samym w sobie, atrybuty defektu mają za zadanie dostarczyć wystarczającej ilości informacji do podjęcia decyzji.

Zgłoszenie defektu pozwalające na podjęcie działania jest:

- kompletne
- zwięzłe
- precyzyjne
- obiektywne

Poza rozwiązaniem defektu, podana informacja powinna pozwalać na trafną klasyfikację, analizę ryzyka oraz udoskonalenie procesów.

7.5 Metryki i zarządzanie incydentami

Informacje o defektach powinny być na tyle dokładne, aby umożliwić monitorowanie postępu testów, analizę gęstości błędów, zbieranie metryk dotyczących liczby błędów znalezionych w stosunku do poprawionych oraz metryk zbieżności (otwarte vs. zamknięte). Dodatkowo informacje te powinny wspierać udoskonalanie procesów poprzez śledzenie występowania defektów w określonych fazach, analizę przyczyny podstawowej oraz identyfikację trendów, które mogą być użyte w działaniach strategicznych mających na celu zmniejszanie ryzyka.

7.6 Komunikowanie incydentów

Zarządzanie incydentami obejmuje również efektywną komunikację, tj. komunikację wolną od oskarżeń, wspierającą zbieranie i interpretowanie obiektywnych informacji. Precyzyjne zgłoszenia incydentów, ich prawidłowa klasyfikacja oraz obiektywność są niezbędne dla utrzymania

profesjonalnych relacji pomiędzy ludźmi zgłaszającymi defekty, a tymi, którzy je usuwają. Testerzy mogą być proszeni o konsultacje w zakresie wagi defektu, a sami powinni dostarczać obiektywnych informacji.

W celu poprawnej priorytetyzacji defektów można organizować dodatkowe spotkania oceniające defekty (ang. *defect triage meetings*). Narzędzia do śledzenia defektów nie powinny być traktowane jako substytut dobrej komunikacji, tak samo jak spotkania oceniające defekty nie powinny być traktowane jako substytut dobrego narzędzia do śledzenia defektów. Chcąc, aby proces zarządzania incydentami był efektywny, należy zadbać zarówno o dobrą komunikację, jak i o odpowiednie wsparcie narzędziowe.

8. Standardy & Udoskonalanie Procesu Testowego

Terminologia

Model Dojrzałości Organizacyjnej (ang. *Capability Maturity Model - CMM*), Zintegrowany Model Dojrzałości Organizacyjnej (ang. *Capability Maturity Model Integration - CMMI*), Model Dojrzałości Testów (ang. *Testing Maturity Model - TMM*), Zintegrowany Model Dojrzałości Testów (ang. *Testing Maturity Model integrated - TMMi*), Usprawnianie Procesu Testowego (ang. *Test Process Improvement - TPI*)

8.1 Wprowadzenie

Wsparcie dla tworzenia i udoskonalania procesu testowego może pochodzić z różnych źródeł. W tym rozdziale przyjęto założenie, że standardy są użytecznym (czasami wymaganym) źródłem informacji dla zagadnień związanych z testowaniem. Wiedza na temat dostępnych standardów oraz ich zastosowania jest celem nauczania dla kierowników testów i testerów. Dostawcy szkoleń powinni zwrócić uwagę słuchaczy na standardy szczególnie związane z nauczanym modułem.

Od momentu wprowadzenia proces testowania powinien być ciągle doskonalony. Podrozdział 8.3 zawiera ogólne uwagi na temat doskonalenia procesu testowania wraz z wprowadzeniem do konkretnych modeli, które mogą zostać użyte. Kierownicy testów powinni zrozumieć całość materiału z tego rozdziału. Istotne jest również, aby analitycy testów i techniczni analitycy testów, jako główne osoby wprowadzające usprawnienia w życie, byli świadomi, jakie modele doskonalenia procesu testowania są dostępne.

8.2 Uznane standardy

W tym syllabusie oraz w syllabusie poziomu podstawowego wymienione są różne standardy. Istnieje wiele standardów dotyczących oprogramowania, np. standardy opisujące:

- cykle życia oprogramowania
- testowanie oprogramowania i metodyki
- zarządzanie konfiguracją oprogramowania
- utrzymanie oprogramowania
- zapewnienie jakości
- zarządzanie projektami
- wymagania
- języki programowania
- interfejsy oprogramowania
- zarządzanie błędami

Celem tego syllabusa nie jest wymienienie lub rekomendacja konkretnych standardów. Testerzy powinni być świadomi tego, jak powstają standardy oraz jak robić z nich użytek w życiu codziennym.

Standardy mogą pochodzić z różnych źródeł:

- międzynarodowych lub z aspiracjami międzynarodowymi
- narodowych, np. narodowych zastosowań międzynarodowych standardów
- dziedzinowych, kiedy międzynarodowe lub narodowe standardy są dostosowywane do potrzeb poszczególnych dziedzin albo są tworzone z myślą o tych dziedzinach

Standardy trzeba wdrażać z rozwagą. Elementy, które podczas wdrażania standardów należy wziąć pod uwagę, są opisane dalej w tym podrozdziale.

8.2.1 Ogólne zagadnienia związane ze standardami

8.2.1.1 Źródła standardów

Standardy są tworzone przez grupy ekspertów i wyrażają mądrość zbiorową. Istnieją dwa główne źródła standardów międzynarodowych: IEEE i ISO. Dostępne są również standardy narodowe i dziedzinowe. Są one równie ważne.

Testerzy powinni być świadomi, które standardy mają zastosowanie w ich środowisku i kontekście pracy, czy to standardy formalne (międzynarodowe, narodowe lub dziedzinowe), czy standardy firmowe i rekomendowane praktyki. Ze względu na fakt, iż standardy zmieniają się, konieczne jest podawanie wersji (daty publikacji) standardu, żeby zapewnić, że została osiągnięta zgodność ze standardem. Kiedy odwołanie do standardu nie zawiera daty publikacji ani wersji, oznacza to odwołanie do najnowszej wersji standardu.

8.2.1.2 Przydatność standardów

Standardy mogą być instrumentem promocji konstruktywnego zapewnienia jakości, które skupia się na minimalizowaniu wprowadzania usterek, w przeciwieństwie do znajdowania ich poprzez testy (analityczne zapewnienie jakości). Nie wszystkie standardy nadają się do zastosowania we wszystkich projektach. Informacja zawarta w standardzie może być użyteczna dla projektu, ale może jemu również zaszkodzić. Postępowanie według standardów będące celem samym w sobie nie pomoże testerowi znaleźć większej liczby defektów w produkcie [Kaner02]. Jednakże standardy mogą stanowić odniesienie i bazę, na której można definiować rozwiązania testowe.

8.2.1.3 Zgodność i konflikty

Niektóre standardy mogą być niespójne z innymi standardami lub nawet podawać sprzeczne definicje. Do użytkownika standardu należy określenie przydatności poszczególnych standardów w jego środowisku i kontekście pracy.

8.2.2 Standardy międzynarodowe

8.2.2.1 ISO

ISO [www.iso.org] jest skrótem od *International Standards Organization* (ostatnio zmienione na *International Organization for Standardization*) i składa się z członków reprezentujących w

poszczególnych krajach organizacje najbardziej reprezentatywne w obszarze standaryzacji. Ta międzynarodowa organizacja ogłosiła wiele standardów przydatnych dla testerów oprogramowania, takich jak:

- ISO 9126:1998, który jest teraz podzielony na następujące standardy i raporty techniczne:
  - *ISO/IEC 9126-1:2001 Software engineering -- Product quality -- Part 1: Quality model*
  - *ISO/IEC TR 9126-2:2003 Software engineering -- Product quality -- Part 2: External metrics*
  - *ISO/IEC TR 9126-3:2003 Software engineering -- Product quality -- Part 3: Internal metrics*
  - *ISO/IEC TR 9126-4:2004 Software engineering -- Product quality -- Part 4: Quality in use metrics*
- *ISO 12207:1995/Amd 2:2004 Systems and Software Engineering -- Software Lifecycle Processes*
- *ISO/IEC 15504[9]-2:2003 Information technology -- Process assessment -- Part 2: Performing an assessment*

Powyższa lista nie wyczerpuje wszystkich standardów ISO, które mogą mieć zastosowanie w środowisku i kontekście testowania.

8.2.2.2 IEEE

IEEE [www.ieee.org] jest skrótem od *Institute of Electrical and Electronics Engineer* - organizacji zawodowej mającej swoją główną siedzibę w USA. Posiada ona oddziały narodowe w ponad stu krajach. Organizacja ta ogłosiła wiele standardów przydatnych dla testerów oprogramowania, takich jak:

- *IEEE 610:1991 IEEE standard computer dictionary* - kompilacja słowników IEEE zawierająca terminologię z zakresu komputerów
- *IEEE 829:1998 IEEE standard for software test documentation*
- *IEEE 1028:1997 IEEE standard for software reviews*
- *IEEE 1044:1995 IEEE guide to classification for software anomalies*

Powyższa lista nie wyczerpuje wszystkich standardów IEEE, które mogą mieć zastosowanie w środowisku i kontekście testowania.

8.2.3 Standardy narodowe

Wiele krajów posiada własne standardy. Niektóre z nich mogą mieć zastosowanie lub być przydatne w testowaniu oprogramowania. Jednym z takich standardów jest brytyjski standard *BS-7925-2:1998 "Software testing. Software component testing"*, który zawiera informacje na temat wielu technik testowania prezentowanych w tym syllabusie:

- podział na klasy równoważności
- analiza wartości brzegowych

[9] Standard ISO 15504 jest znany również jako SPICE.

- testowanie przejść pomiędzy stanami
- tworzenie grafów przyczynowo-skutkowych
- testowanie instrukcji
- testowanie rozgałęzień/decyzji
- testowanie warunków
- testowanie losowe

Standard BS-7925-2 zawiera również opis procesu testów komponentów.

8.2.4 Standardy dziedzinowe

Standardy używane są w wielu branżach. W niektórych gałęziach przemysłu standardy są dostosowywane do specyficznych potrzeb danej dziedziny. Poniżej wymieniono aspekty związane z tymi standardami, które mogą wydać się interesujące osobom zajmującym się testowaniem oprogramowania, jakością oprogramowania oraz wytwarzaniem oprogramowania.

8.2.4.1 Lotnictwo

Standardem stosowanym w lotnictwie cywilnym jest *RTCA DO-178B/ED 12B "Software Considerations in Airborne Systems and Equipment Certification".* Standard ten odnosi się również do oprogramowania używanego w produkcji (lub weryfikacji) oprogramowania używanego w awionice. Dla oprogramowania lotniczego amerykańska organizacja *Federal Aviation Administration (FAA)* oraz międzynarodowe jednostki *Joint Aviation Authorities (JAA)* ustanowiły poziomy pokrycia strukturalnego, które zależą od poziomu krytyczności testowanego oprogramowania:

Poziom krytyczności	**Potencjalne skutki awarii**	**Wymagany poziom pokrycia strukturalnego**
A Katastroficzny	Uniemożliwia kontynuowanie bezpiecznego lotu i lądowanie	pokrycie warunków znaczących, decyzji i instrukcji
B Niebezpieczny / Bardzo wysoki	Duże zmniejszenie marginesów bezpieczeństwa lub możliwości funkcjonalnych. Nie można polegać na załodze, że wykona swoje zadania dokładnie lub skutecznie. Poważne lub śmiertelne obrażenia małej liczby pasażerów.	pokrycie decyzji i instrukcji
C Wysoki	Znaczące zmniejszenie marginesów bezpieczeństwa. Znaczące zwiększenie obciążenia załogi. Dyskomfort pasażerów oraz możliwe obrażenia.	pokrycie instrukcji
D Niski	Niewielkie zmniejszenie marginesów bezpieczeństwa lub możliwości funkcjonalnych. Niewielkie zwiększenie	brak

	obciążenia załogi. Niewielkie utrudnienia dla pasażerów.	
E Żaden	Brak wpływu na możliwości statku powietrznego. Brak wpływu na obciążenie załogi.	brak

W zależności od poziomu krytyczności oprogramowania certyfikowanego na potrzeby lotnictwa cywilnego, musi zostać osiągnięty określony poziom pokrycia.

8.2.4.2 Przemysł kosmiczny

Niektóre gałęzie przemysłu dostosowują standardy do swoich specyficznych dziedzin. Tak jest w przypadku przemysłu kosmicznego w ECSS (*European Cooperation on Space Standardization*) [www.ecss.org]. W zależności od poziomu krytyczności oprogramowania ECSS rekomenduje metody i techniki, które są zgodne z syllabusem poziomu podstawowego ISTQB® i syllabusem poziomu zaawansowanego ISTQB®, w tym:

- *SFMECA - Software Failure Modes, Effects and Criticality Analysis*
- *SFTA - Software Fault Tree Analysis*
- *HSIA - Hardware Software Interaction Analysis*
- *SCCFA - Software Common Cause Failure Analysis*

8.2.4.3 Żywność i leki

Dla systemów medycznych podlegających ustawie *Title 21 CFR Part 820*, amerykańska *Food and Drug Administration (FDA)* rekomenduje użycie pewnych funkcjonalnych i strukturalnych technik testowania. Strategie i techniki rekomendowane przez FDA są zgodne z tymi zalecanymi przez syllabus poziomu podstawowego ISTQB® i syllabus poziomu zaawansowanego ISTQB®.

8.2.5 Inne standardy

Liczba standardów obowiązujących w różnych gałęziach przemysłu jest bardzo duża. Niektóre z nich zostały przystosowane do potrzeb poszczególnych dziedzin, czy gałęzi przemysłu, inne natomiast mają zastosowanie tylko do pewnych zadań lub podają wyjaśnienia, jak wybrany standard powinien zostać wdrożony.

Na testerze spoczywa obowiązek poznania różnych standardów (włączając w to standardy firmowe, najlepsze praktyki, itp.), które mają zastosowanie w jego dziedzinie, gałęzi przemysłu lub kontekście. Czasami są one wymieniane, wraz z hierarchią zastosowań, w określonych kontraktach. Do zadań kierownika testów należy posiadanie wiedzy, z którymi standardami należy być zgodnym, oraz utrzymanie tej zgodności.

8.3 Usprawnianie procesu testowania

Tak jak testowanie służy do doskonalenia oprogramowania, procesy jakościowe są wybierane i stosowane w celu doskonalenia procesu wytwarzania oprogramowania (a w konsekwencji i samego oprogramowania, które jest wynikiem tego procesu). Doskonaleniu mogą również podlegać procesy testowania. Dostępne są różne sposoby i metody usprawniania testowania oprogramowania i testowania systemów zawierających oprogramowanie. Celem tych metod jest usprawnienie procesu (a tym samym również produktów tego procesu) przez dostarczenie wytycznych i wskazanie obszarów doskonalenia.

Testowanie często stanowi dużą część kosztów projektu. Jednak niewiele uwagi poświęca się samemu procesowi testowania w różnych modelach doskonalenia procesu produkcji oprogramowania, takich jak CMMI (patrz niżej).

Modele doskonalenia testowania takie jak Model Dojrzałości Testów (ang. *Test Maturity Model - TMM*), *Systematic Test and Evaluation Process* (STEP), krytyczne procesy testowania (ang. *Critical Testing Processes* (CTP)) oraz Usprawnianie Procesu Testowego (ang. *Test Process Improvement - TPI*) powstały, aby uzupełnić tę lukę. TPI i TMM podają metryki (ang. *benchmark*), które mogą być wykorzystywane w porównaniach pomiędzy organizacjami. Obecnie istnieje wiele takich modeli doskonalenia. Oprócz opisanych w tym rozdziale powinno się wziąć pod uwagę jeszcze *Test Organization Maturity (TOM), Test Improvement Model (TIM)* oraz *Software Quality Rank (SQR).* Należy przy tym również wspomnieć, że istnieje duża liczba obecnie używanych modeli regionalnych. Testerzy powinni zapoznać się ze wszystkimi dostępnymi modelami, aby wybrać ten najbardziej im pasujący.

Przedstawienie model w niniejszym rozdziale nie ma na celu rekomendowania wybranych modeli, jest raczej próbą ukazania, jak modele działają i co się na nie składa.

8.3.1 Wstęp do doskonalenia procesów

Pojęcie usprawniania procesów odnosi się tak samo do procesu wytwarzania oprogramowania, jak i do procesu testowania. Uczenie się na własnych błędach umożliwia doskonalenie procesów produkcji i testowania oprogramowania w organizacjach. Cykl Deminga (zaplanuj – wykonaj – sprawdź – zastosuj ang. *Plan, Do, Check, Act*) jest używany od wielu lat i nadal znajduje zastosowanie w usprawnianiu procesu testowania.

Podstawowym założeniem przy doskonaleniu procesu jest przekonanie, że jakość systemu jest w dużym stopniu zależna od jakości procesu, który doprowadził do jego wytworzenia. Wyższa jakość oprogramowania zmniejsza ilość zasobów potrzebnych do jego utrzymania i w ten sposób zwiększa ilość czasu dostępnego na stworzenie większej liczby, lepszych rozwiązań w przyszłości.

Modele procesów wskazują miejsce rozpoczęcia procesu doskonalenia przez pomiar dojrzałości istniejących procesów w organizacji. Model dostarcza również wytycznych dla doskonalenia procesów w organizacji bazując na wynikach oceny procesów.

Po ocenie procesu następuje ustalenie jego możliwości, co staje się bodźcem do jego doskonalenia. Po tym może nastąpić ponowna ocena procesu, której celem będzie pomiar efektywności przeprowadzonych usprawnień.

8.3.2 Typy doskonalenia procesów

Istnieją dwa typy modeli: model referencyjny procesu oraz model referencyjny zawartości.

1. Model referencyjny procesu stanowi szkielet używany podczas dokonywania oceny, stosowany w celu porównania możliwości organizacji z modelem i dokonania oceny organizacji według modelu.
2. Model referencyjny zawartości używany jest do udoskonalenia procesu po dokonaniu jego oceny.

Niektóre modele mogą zawierać obie części, podczas gdy inne - tylko jedną z nich.

8.4 Doskonalenie procesu testowania

W przemysle IT zaczęto stosować modele doskonalenia procesów, ponieważ dążono do osiągnięcia wyższego poziomu dojrzałości i profesjonalizmu. Modele, które stały się standardami w przemyśle, pomagają w rozwinięciu metryk i miar, które mogą być używane do porównywania organizacji. Modele oparte na poziomach, takie jak TMMi oraz CMMI, dają podstawę do porównania różnych firm i organizacji. Modele ciągłe pozwalają organizacji na zajęcie się problemami o największym priorytecie, dając więcej swobody w wyborze kolejności wdrażania usprawnień. Potrzeba usprawniania procesów spowodowała powstanie kilku modeli w przemyśle testowym – m.in. STEP, TMMi, TPI i CTP, które są omówione w kolejnych podrozdziałach.

Wszystkie cztery wymienione modele oceny procesów pozwalają organizacji na ustalenie, w jakim stanie są jej procesy testowe. TMMi oraz TPI, po wykonaniu oceny procesów, pokazują drogę do ich udoskonalenia. Natomiast STEP i CTP wskazują miejsca, których udoskonalenie najbardziej się opłaca a już organizacji pozostawiają wypracowanie najlepszej drogi dojścia do celu.

Po ustaleniu, że procesy testowe powinny zostać przejrzane i udoskonalone, należy wykonać następujące kroki:

- Rozpocznij (***I**nitiate*)
- Zmierz (***M**easure*)
- Ustal priorytety i zaplanuj (***P**rioritize and Plan*)
- Zdefiniuj i przedefiniuj (*Define and **R**edefine*)
- Wykonaj (***O**perate*)
- Sprawdź (***V**alidate*)
- Rozwijaj (***E**volve*)

Rozpocznij

W tym kroku potwierdza się interesariuszy, cele, zakres i pokrycie dla doskonalenia procesów. Tutaj również wybiera się model doskonalenia procesów, który zostanie użyty. Model może zostać wybrany z modeli publicznie dostępnych albo opracowany samodzielnie.

Przed rozpoczęciem doskonalenia procesów należy zdefiniować kryteria sukcesu oraz wdrożyć metodę ich pomiaru dla całego procesu doskonalenia.

Zmierz

Proces ocenia się według uzgodnionego podejścia, co w konsekwencji pozwala zidentyfikować listę usprawnień.

Ustal priorytety i zaplanuj

Potencjalne usprawnienia są szeregowane według priorytetów. Kolejność na liście może wynikać ze stopy zwrotu z inwestycji, ryzyka, dopasowania do strategii organizacji, mierzalnych ilościowych lub jakościowych korzyści.

Po uszeregowaniu elementów listy według priorytetów, tworzony jest plan budowy i wdrożenia usprawnień.

Zdefiniuj i przedefiniuj

Na podstawie zidentyfikowanych wymagań na doskonalenie procesów, definiuje się nowe potrzebne procesy, bądź uaktualnia te istniejące, a następnie przygotowuje je do wdrożenia.

Wykonaj

Po opracowaniu usprawnień następuje ich wdrożenie. Wdrożenie może pociągać za sobą konieczność przeprowadzenia niezbędnych szkoleń lub zapewnienia potrzebnego wsparcia (mentoringu); może mieć charakter pilotażowy lub kompleksowy.

Sprawdź

Po pełnym wdrożeniu usprawnień procesów konieczne jest sprawdzenie, czy osiągnięto korzyści, które zostały uzgodnione przed rozpoczęciem doskonalenia procesów. Istotne jest również spełnienie zaplanowanych kryteriów sukcesu.

Rozwijaj

W zależności od używanego modelu doskonalenia procesów, w tym kroku rozpoczyna się monitorowanie kolejnego poziomu dojrzałości i podejmowana jest decyzja o ponownym rozpoczęciu procesu doskonalenia lub zakończeniu działań.

Użycie modeli oceny jest często spotykaną praktyką, która zapewnia ustandaryzowane podejście do doskonalenia procesów testowania z wykorzystaniem wypróbowanych i pewnych metod. Doskonalenie testowania można również skutecznie przeprowadzić bez modeli, a przy użyciu na przykład metod analitycznych oraz spotkań podsumowujących.

8.5 Doskonalenie procesu testowania z użyciem modelu TMM

Model Dojrzałości Testów (ang. *Testing Maturity Model – TMM*) określa pięć poziomów dojrzałości (ang. *maturity levels*) i jest pomyślany jako uzupełnienie modelu CMM. Dla każdego z poziomów zdefiniowano obszary procesów (ang. *process areas*), które muszą być całkowicie wdrożone zanim organizacja osiągnie kolejny poziom dojrzałości (reprezentacja etapowa). TMM zawiera zarówno model referencyjny procesów, jak i zawartości.

W TMM występują następujące poziomy dojrzałości:

Poziom 1: Inicjacja *(Initial)*

> Poziom inicjacji reprezentuje stan, kiedy nie istnieje formalnie udokumentowany lub zdefiniowany proces testowania. Testy są tworzone ad hoc po kodowaniu, a testowanie jest postrzegane jako równoważne z debagowaniem. Za cel testów uważa się wykazanie, że oprogramowanie działa.

Poziom 2: Definicja *(Definition)*

> Poziom drugi może zostać osiągnięty po ustanowieniu celów oraz polityki testowania, wdrożeniu procesu planowania testów i zaimplementowaniu podstawowych metod i technik testowania.

Poziom 3: Integracja *(Integration)*

> Poziom trzeci zostaje osiągnięty, gdy proces testowania zostaje zintegrowany z procesem życia oprogramowania oraz jest udokumentowany w postaci formalnych standardów, procedur i metodyk. W organizacji powinna istnieć wydzielona funkcja testowania oprogramowania, która może być kontrolowana i monitorowana.

Poziom 4: Zarządzanie i pomiar *(Management & Measurement)*

> Poziom czwarty zostaje osiągnięty, gdy testowanie może być skutecznie zmierzone, zarządzane i dostosowywane do potrzeb konkretnych projektów.

Poziom 5: Optymalizacja *(Optimization)*

Ten ostatni poziom reprezentuje stan dojrzałości procesu testowania, kiedy dane z testowania mogą być użyte w celu zapobiegania usterkom i kiedy organizacja skupia się na optymalizacji wdrożonego procesu.

Żeby osiągnąć poszczególne poziomy dojrzałości muszą zostać osiągnięte zdefiniowane cele dojrzałości (ang. *maturity goals*) oraz cele cząstkowe dojrzałości (ang. *maturity sub-goals*). Są one definiowane w odniesieniu do działań, zadań oraz odpowiedzialności i oceniane przez określone "perspektywy": kierownika, programisty/testera oraz klienta/użytkownika. Więcej informacji na temat TMM znajduje się w [Burnstein03].

TMMi Foundation [zobacz www.tmmifoundation.org] zdefiniowała następcę TMM – model TMMi. TMMi jest szczegółowym modelem doskonalenia procesu testowania bazującym na TMM i doświadczeniach z jego stosowania, rozwijanym przez *Illinois Institute of Technology*. TMMi pozycjonowany jest jako uzupełnienie modelu CMMI.

Struktura TMMi w większości bazuje na strukturze CMMI (np. obszary procesów – ang. *process areas*, cele ogólne – ang. *generic goals*, ogólne praktyki – ang. *generic practices*, cele szczegółowe – ang. *specific goals*, szczegółowe praktyki – ang. *specific practices*).

8.6 Doskonalenie procesu testowania z użyciem modelu TPI

W przeciwieństwie do TMM, TPI stosuje podejście ciągłe do doskonalenia procesów, a nie oparte na reprezentacji etapowej.

Plan optymalizacji procesu testowania, tak jak został przedstawiony w [Koomen99], składa się ze zbioru kluczowych obszarów (ang. *key areas*) umieszczonych na czterech podstawach: cyklu życia, organizacji, infrastrukturze i narzędziach oraz technikach. Obszary kluczowe są oceniane w skali od A do D, gdzie A jest oceną najniższą. Możliwe jest, że bardzo niedojrzałe obszary kluczowe nie uzyskają nawet podstawowej noty A. Niektóre obszary (np. szacowanie i planowanie) mogą być oceniane wyłącznie w skali A lub B, a inne (np. metryki) - na poziomie A, B, C lub D.

Ocena poziomu dla poszczególnych kluczowych obszarów odbywa się przy pomocy zdefiniowanych w modelu TPI punktów kontrolnych (ang. *checkpoints*). Gdy, na przykład, wszystkie punkty kontrolne dla poziomów A i B w obszarze Raportowanie są spełnione, to obszar ten uzyskuje poziom B.

TPI definiuje zależności pomiędzy różnymi obszarami kluczowymi i poziomami. Zależności te zapewniają równomierny rozwój procesu testowania. Nie jest możliwe, na przykład, uzyskanie poziomu A w obszarze Metryki, bez uzyskania poziomu A w obszarze Raportowanie (jaki byłby pożytek z metryk, jeżeli nie podlegają one raportowaniu). Wykorzystanie zależności jest w TPI opcjonalne.

TPI jest przede wszystkim modelem referencyjnym procesu.

Macierz Dojrzałości Testowania (ang. *Test Maturity Matrix*) mapuje poziomy (A, B, C i D) każdego z kluczowych obszarów określając w ten sposób ogólny poziom dojrzałości procesu testowania. Tymi ogólnymi poziomami są następujące poziomy:

- Kontrolowany *(Controlled)*
- Efektywny *(Efficient)*
- Optymalizowany *(Optimizing)*

Podczas oceny procesów według modelu TPI, do ustalenia poziomu dojrzałości procesów używane są zarówno metryki ilościowe, jak i wywiady jakościowe.

8.7 Doskonalenie procesu testowania z użyciem modelu CTP

Tak jak to zostało opisane w [Black02], podstawowym założeniem modelu o nazwie Krytyczny Proces Testowania (ang. *Critical Test Process - CTP*) jest to, że niektóre procesy testowe są krytyczne. Jeżeli zostaną one wykonane poprawnie, przyczynią się do sukcesu zespołów testowych. I odwrotnie, jeżeli zostaną przeprowadzone niestarannie, to nawet utalentowani testerzy czy kierownicy testów mają małe szanse na sukces. Model identyfikuje dwanaście krytycznych procesów testowych.

CTP jest głównie modelem referencyjnym zawartości.

Model CTP jest wrażliwy na kontekst i pozwala na dostosowywanie go, włączając w to:

- identyfikację konkretnych problemów
- rozpoznanie atrybutów dobrych procesów
- dobór porządku i priorytetów wdrażania usprawnień procesu

Model CTP udaje się dostosować do dowolnego modelu cyklu życia oprogramowania.

Doskonalenie procesu w oparciu o model CTP rozpoczyna się od oceny istniejącego procesu testowego. W ramach oceny identyfikuje się mocne i słabe procesy oraz rekomenduje kolejność wprowadzania usprawnień bazując na potrzebach organizacji. Chociaż procesy oceniania różnią się w zależności od konkretnego kontekstu, w którym są wykonywane, przeważnie sprawdza się następujące metryki:

- odsetek wykrytych defektów
- zwrot z inwestycji w testowanie
- pokrycie wymagań oraz pokrycie ryzyka
- narzut na tworzenie wersji testowej
- stopień odrzucenia zgłoszeń defektów

Następujące czynniki jakościowe są zwykle oceniane podczas CTP:

- rola i skuteczność zespołu testowego
- użyteczność planu testów

- umiejętności zespołu testowego dotyczące testowania, wiedzy dziedzinowej i technologii
- użyteczność zgłoszeń defektów
- użyteczność raportu wyników testów
- użyteczność i bilans zarządzania zmianą

W sytuacji, gdy podczas oceny zostaną zidentyfikowane obszary wymagające usprawnień, wdrażane są plany doskonalenia. Model zawiera generyczne plany doskonalenia dla każdego z krytycznych procesów, ale od zespołu oceniającego oczekuje się mocnego dostosowania planów do potrzeb.

8.8 Doskonalenie procesu testowania z użyciem modelu STEP

Proces Systematycznego Testowania i Oceny (ang. *Systematic Test and Evaluation Process - STEP*) tak jak CTP i w przeciwieństwie do TMMi i TPI, nie wymaga wdrażania usprawnień w określonym porządku.

Podstawowe założenia tej metodyki są następujące:

- strategia testowania oparta jest na wymaganiach
- testowanie rozpoczyna się już na początku cyklu życia oprogramowania
- testy wykorzystuje się jako modele wymagań i użycia
- projektowanie testaliów wyprzedza projektowanie oprogramowania
- defekty są wykrywane wcześnie lub udaje się im kompletnie zapobiec
- defekty poddawane są systematycznej analizie
- testerzy i programiści pracują wspólnie

STEP jest w głównej mierze modelem referencyjnym zawartości.

Metodyka STEP bazuje na idei, że testowanie jest czynnością cyklu życia oprogramowania, która rozpoczyna się podczas specyfikacji wymagań i trwa aż do wycofania systemu. Metodyka STEP kładzie nacisk na podejście "najpierw testuj, potem koduj" przez wykorzystanie strategii testowania opartej na wymaganiach. Ma to na celu zapewnienie, że podczas wczesnego tworzenia przypadków testowych, specyfikacja wymagań zostanie sprawdzona, a nastąpi to jeszcze przed projektowaniem i kodowaniem. Metodyka identyfikuje i skupia się na doskonaleniu trzech głównych faz testowania:

- planowanie *(planning)*
- zbieranie *(acquisition)*
- pomiar *(measurement)*

Podczas procesu oceny według STEP wykorzystywane są metryki ilościowe oraz wywiady jakościowe. Używanymi metrykami są:

- status testów w czasie
- pokrycie wymagań testów lub ryzyka
- trendy defektów, a w tym wykrycie, przypisanie wagi oraz grupowanie
- gęstość defektów

- skuteczność usuwania defektów
- odsetek wykrywanych defektów
- fazy wprowadzania, wykrywania i usuwania defektów
- koszt testowania wyrażony w czasie, pracochłonności i pięniądzach

Czynniki jakościowe to:

- zdefiniowane wykorzystanie procesu testowania
- satysfakcja klienta

W niektórych przypadkach model oceny STEP jest łączony z modelem dojrzałości TPI.

8.9 Capability Maturity Model Integration, CMMI

Zintegrowany Model Dojrzałości Organizacyjnej (ang. *Capability Maturity Model Integration - CMMI*) może być wdrażany w dwóch różnych podejściach lub reprezentacjach: etapowej lub ciągłej. W reprezentacji etapowej jest pięć "poziomów dojrzałości", z których każdy bazuje na obszarach procesów wdrożonych na niższych poziomach. W reprezentacji ciągłej organizacja może skupić się na swoich głównych potrzebach doskonalenia niezależnie od hierarchii poziomów.

Reprezentacja etapowa jest obecna w CMMI, głównie w celu utrzymania zgodności z CMM, a reprezentacja ciągła jest generalnie uważana za bardziej elastyczną.

Obszar procesów Weryfikacji i Walidacji w CMMI obejmuje zarówno testy dynamiczne, jak i statyczne.

9. Narzędzia testowe i automatyzacja testów

Terminologia

narzędzie do debagowania, narzędzie do analizy dynamicznej, emulator, narzędzie do posiewu usterek, narzędzie do testowania linków, testowanie oparte o słowa kluczowe, narzędzie do testów wydajnościowych, symulator, analizator statyczny, narzędzie do wykonywania testów, narzędzie do zarządzania testami, wyrocznia testowa

9.1 Wprowadzenie

Rozdział ten rozszerza materiał zawarty w syllabusie poziomu podstawowego zajmując się najpierw ogólnymi koncepcjami związanymi z narzędziami testowymi, a później bardziej szczegółowo rozważając niektóre z narzędzi.

Chociaż część z omówionych tu koncepcji może być bardziej przydatna dla kierownika testów, analityka testów, czy technicznego analityka testów, podstawowe ich zrozumienie wymagane jest od wszystkich zawodowych testerów. Wiedzę tę można później pogłębiać tam, gdzie jest to potrzebne.

Narzędzia mogą być pogrupowane na wiele sposobów, włączając w to grupowanie po użytkownikach, takich jak kierownicy testów, analitycy testów, czy techniczni analitycy testów. W niniejszym rozdziale został użyty ten szczególny podział, który odzwierciedla układ modułów w syllabusie. Ogólnie rzecz biorąc narzędzia omawiane w poszczególnych podrozdziałach odnoszą się głównie do konkretnych modułów, chociaż niektóre z nich (np. narzędzia do zarządzania testami) mogą mieć szersze zastosowanie. W tych przypadkach dostawca szkolenia poda przykłady ich użycia w określonym kontekście.

9.2 Koncepcje związane z narzędziami testowymi

Narzędzia testowe mogą znacznie poprawić efektywność i dokładność testowania, ale tylko wtedy, gdy odpowiednie narzędzia zostaną zastosowane w odpowiedni sposób. Narzędziami testowymi trzeba zarządzać tak, jak każdym aspektem dobrze prowadzonej organizacji testów. Automatyzacja testów często jest uważana za synonim wykonywania testów, ale większość ręcznej pracy przy testach poddaje się różnym formom automatyzacji. Oznacza to, że większość obszarów testowania może zostać do pewnego stopnia zautomatyzowana, pod warunkiem posiadania odpowiednich narzędzi.

Każda z wersji narzędzi testowych, skryptów testowych oraz sesji testowej powinna podlegać (tak jak każda podstawa testów) zarządzaniu konfiguracją i być powiązana z wersją oprogramowania, dla której została użyta. Każde z narzędzi testowych jest ważnym elementem testaliów i powinno być odpowiednio zarządzane, to znaczy, że należy:

- stworzyć architekturę przed utworzeniem narzędzia testowego

- zapewnić odpowiednie zarządzanie konfiguracją skryptów, wersji narzędzi, łat (ang. *patches*) itp., włączając w to informację o wersji
- utworzenie i utrzymywanie bibliotek (reużycie podobnych koncepcji w ramach przypadków testowych), udokumentowanie wdrożenia narzędzi testowych (np. proces wykorzystywania narzędzia przez organizację)
- planowanie pod kątem przyszłości przez strukturalizowanie przypadków testowych dla przyszłego rozwoju np. przez takie ich utworzenie, żeby były rozszerzalne i utrzymywalne

9.2.1 Korzyści finansowe i ryzyko narzędzi testowych i automatyzacji

W każdym przypadku powinna zostać przeprowadzona analiza kosztów i zysków, przy czym powinna ona wykazać znaczący zwrot z inwestycji. Analiza kosztów i zysków powinna zawierać porównanie następujących grup kosztów związanych z pracą ręczną i wspomaganą narzędziami (roboczogodziny wyrażone jako koszt, koszty bezpośrednie, koszty jednorazowe i cykliczne):

- koszty początkowe
  - przyswajanie wiedzy (krzywa uczenia się narzędzia)
  - ewaluacja (porównanie narzędzi), tam gdzie jest to potrzebne
  - integracja z innymi narzędziami
  - wstępne koszty nabycia, adaptacji lub rozwoju narzędzia
- koszty cykliczne
  - koszt posiadania narzędzia (utrzymanie, opłaty licencyjne, opłaty serwisowe, utrzymanie poziomu wiedzy)
  - przenaszalność
  - dostępność i zależności (jeżeli ich brakuje)
  - ciągła ocena kosztów
  - doskonalenie jakości w celu zapewnienia optymalnego użycia wybranych narzędzi

Uzasadnienia biznesowe oparte jedynie o projekty pilotażowe często pomijają ważne elementy kosztów, takie jak koszt utrzymania, uaktualniania i rozszerzania skryptów testowych przy zmianach systemu. Trwałość przypadków testowych mówi o tym, jak długo przypadek testowy pozostaje aktualny bez modyfikacji. Czas potrzebny na zaimplementowanie pierwszej wersji automatycznego skryptu testowego często znacznie przekracza czas ręcznego ich wykonania, ale umożliwia tworzenie następnych podobnych skryptów dużo szybciej i łatwe rozszerzanie zbiorów dobrych przypadków testowych. Dodatkowo, podczas używania automatyzacji po fazie implementacji znacząco polepszy się pokrycie testowe i efektywność testów. Uzasadnienie biznesowe wdrażania narzędzi testowych musi mieć oparcie w długoterminowych zyskach biznesowych.

Celowość automatyzacji musi być przeanalizowana dla każdego z przypadków testowych na każdym z poziomów testowania. Wiele projektów automatyzacji opiera się na implementacji już gotowych ręcznych przypadków testowych bez rozpatrzenia, czy automatyzacja konkretnych przypadków testowych przyniesie jakiekolwiek korzyści. Najprawdopodobniej każdy zbiór przypadków testowych (zestaw testowy) będzie zawierał testy ręczne, półautomatyczne i zautomatyzowane.

Oprócz tematów omówionych w syllabusie poziomu podstawowego powinny zostać rozważone następujące aspekty:

Dodatkowe korzyści:

- czas wykonania automatycznych testów staje się bardziej przewidywalny
- jeżeli testy są zautomatyzowane, testowanie regresywne i walidacja usterek w końcowej fazie projektu jest szybsza i bezpieczniejsza
- używanie narzędzi do automatyzacji może podnieść status i wspomóc rozwój techniczny testerów i zespołu testowego
- automatyzacja może zostać wykorzystana w równoległym, iteracyjnym i i przyrostowym rozwoju oprogramowania, aby zapewnić lepsze testowanie regresywne każdej wersji
- pokrycie pewnych typów testów, które nie może zostać osiągnięte ręcznie (np. testy wydajnościowe i testy niezawodności)

Dodatkowe obszary ryzyka:

- niekompletne lub nieprawidłowe testy ręczne zostają zautomatyzowane bez zmian
- trudności w utrzymaniu testaliów, gdyż konieczne jest wiele modyfikacji, gdy oprogramowanie poddawane testom się zmieni
- następuje zmniejszenie zaangażowania testerów w wykonanie testów, ponieważ wykonywane są tylko zautomatyzowane, skrypty testowe, co może zmniejszyć wykrywalność defektów

9.2.2 Strategie narzędzi testowych

Narzędzia testowe zwykle są przeznaczone do wykorzystania w więcej niż jednym projekcie. W zależności od wielkości inwestycji i długości projektu, w danym projekcie odpowiedni zwrot z inwestycji może nie zostać osiągnięty, ale mógłby być osiągnięty w kolejnych wersjach oprogramowania. Na przykład, biorąc pod uwagę, że faza utrzymania często wymaga wielu testów (dla każdej poprawki należy wykonać duży zestaw testów regresywnych), może opłacać się automatyzacja testów systemu, który jest w tej fazie pod warunkiem, że długość życia systemu jest wystarczająca. Innym przykładem jest to, że łatwo jest się pomylić wykonując testy ręczne (np. robiąc literówki), więc korzystne może być zautomatyzowanie wprowadzania danych oraz porównania danych wyjściowych z danymi z wyroczni (porównanie wyniku rzeczywistego z oczekiwanym).

Firmy, które korzystają i są zależne od wielu narzędzi (w wielu fazach i dla różnych celów), powinny posiadać długoterminową strategię użycia narzędzi testowych wspierającą podejmowanie decyzji wdrażania i wycofywania różnych wersji narzędzi oraz wsparcia dla nich. W większych firmach, intensywnie korzystających z narzędzi, powinny istnieć wytyczne co do zakupu narzędzi, strategii oraz użycia paradygmatów narzędzi i języków skryptowych.

9.2.3 Integracja i wymiana informacji pomiędzy narzędziami

Zwykle w procesie testowania (i rozwoju oprogramowania) używane jest więcej niż jedno narzędzie. Weźmy przykład firmy używającej równolegle narzędzi do analizy statycznej, do zarządzania testami i raportowania, do zarządzania konfiguracją, do zarządzania incydentami oraz do wykonywania testów. Ważne jest rozważenie, czy te narzędzia mogą wymieniać między sobą informacje w sposób dający korzyści. Na przykład, byłoby przydatne gdyby status wszystkich testów był raportowany

wprost do narzędzia zarządzającego testami, by umożliwić natychmiastowe uaktualnianie postępu testów oraz śledzenie powiązań pomiędzy wymaganiami a konkretnym przypadkiem testowym. Trzymanie skryptów testowych zarówno w bazie zarządzania testami, jak i w systemie do zarządzania konfiguracją jest bardziej pracochłonne i narażone na błędy. Jeżeli tester chce utworzyć raport incydentu w środku wykonywania przypadku testowego, narzędzia do zarządzania testami i do śledzenia defektów muszą być zintegrowane. Chociaż narzędzie do analizy statycznej może nie być połączone z innymi narzędziami, byłoby dużo wygodniej, gdyby mogło raportować incydenty, ostrzeżenia i wyniki analizy wprost do systemu zarządzania testami.

Nabycie zestawu narzędzi testowych od jednego dostawcy nie oznacza automatycznie, że pracują one w ten sposób, ale jest to rozsądnym wymaganiem. Należy ocenić wszystkie aspekty, od kosztu automatyzacji wymiany informacji, po ryzyko fałszowania lub utraty informacji przy przenoszeniu ręcznym, zakładając że organizacja może sobie pozwolić na pracę, jakiej to będzie wymagać.

Nowe koncepcje, takie jak zintegrowane środowiska deweloperskie (np. *Eclipse*) mają na celu ułatwienie integracji i używania różnych narzędzi w tym samym środowisku poprzez udostępnienie wspólnego interfejsu dla narzędzi deweloperskich i testowych. Dostawca narzędzia może stać się "zgodny" z *Eclipse* przez utworzenie wtyczki do *Eclipse*, co nada mu ten sam wygląd i sposób działania, jaki ma każde inne narzędzie. Daje to dużą korzyść dla użytkownika. Należy zauważyć, że podobieństwo interferjsów narzędzi użytkownika nie oznacza automatycznie ich integracji i wymiany informacji między nimi.

9.2.4 Języki automatyzacji: skrypty, języki skryptowe

Aby lepiej zaimplementować lub rozwinąć warunki lub przypadki testowe, czasem używane są skrypty i języki skryptowe. Na przykład podczas testowania aplikacji webowych, można użyć skryptu do pominięcia interfejsu użytkownika i lepszego przetestowania samego API. Innym przykładem może być automatyzacja testów interfejsu użytkownika, w celu wprowadzenia wszystkich kombinacji danych wejściowych, co byłoby niewykonalne przy użyciu testów ręcznych.

Możliwości języków skryptowych bardzo się różnią. Należy zauważyć, że języki skryptowe mogą być zarówno normalnymi językami programowania, jak i bardzo wyspecjalizowanymi standardami notacji (np. TTCN-3).

9.2.5 Wyrocznie testowe

Wyrocznie testowe używane są głównie do ustalenia wyników oczekiwanych. Jako takie wykonują więc te same funkcje, co testowane oprogramowanie i są rzadko dostępne. Jednakże mogą być używane w sytuacji, gdy stary system jest zastępowany nowym, realizującym te same funkcje. W takich przypadkach stary system może być wyrocznią testową. Wyrocznie mogą być również używane, gdy istnieje problem z wydajnością dostarczanego systemu. Do generowania oczekiwanych wyników dla testów funkcjonalnych dostarczanego oprogramowania o wysokiej wydajności, można zbudować lub wykorzystać wyrocznę o niskiej wydajności.

9.2.6 Wdrażanie narzędzi testowych

Każde z narzędzi testowych samo w sobie jest programem i może posiadać programowe lub sprzętowe zależności. Narzędzie powinno być udokumentowane i przetestowane niezależnie od tego, czy zostało kupione, zaadaptowane, czy wytworzone lokalnie. Niektóre narzędzia są bardziej zintegrowane ze środowiskiem, natomiast inne lepiej pracują pojedynczo.

W przypadkach, gdy testowany system działa na własnym sprzęcie, systemie operacyjnym, oprogramowaniu osadzonym lub używa niestandardowej konfiguracji, może być konieczne utworzenie (rozwój) narzędzi lub dostosowanie narzędzi do konkretnego środowiska. Zawsze należy przeprowadzić analizę korzyści i kosztów, w której należy ująć zarówno koszty implementacji, jak i długoterminowego utrzymania.

Podczas wdrożenia narzędzi do automatyzacji testów nie zawsze dobrym rozwiązaniem jest automatyzacja przypadków testowych w takiej formie, w jakiej już istnieją, ale raczej przeprojektowanie ich tak, żeby się lepiej nadawały do automatyzacji. Obejmuje to zmianę formatowania przypadków testowych, rozważenie reużycia wzorców, rozszerzenie wejścia przez użycie zmiennych, zamiast ustalonych wartości oraz wykorzystanie zalet narzędzia testowego, które może trawersować, powtarzać, zmieniać kolejność w połączeniu z lepszą analizą i raportowaniem. W przypadku wielu narzędzi do automatyzacji testów, do stworzenia skutecznych i efektywnych skryptów testowych konieczne jest posiadanie umiejętności programistycznych. Często zdarza się, że duże zestawy testów są bardzo trudne do aktualizacji i zarządzania, o ile nie zostały zaprojektowane z dostateczną uwagą. Do pełnego wykorzystania zalet narzędzi wskazane są odpowiednie szkolenia z narzędzi testowych oraz technik programowania i projektowania.

Nawet jeżeli ręczne przypadki testowe zostały zautomatyzowane, bardzo ważne jest wykonywanie ich co jakiś czas manualnie, aby przypomnieć sobie, jak dany test działa i sprawdzić, czy działa poprawnie.

Kiedy narzędzie zaczyna być szerzej używane i liczba skryptów rośnie, może zajść potrzeba wykorzystania funkcji oferowanych przez inne narzędzia. Nie zawsze jest to możliwe, ponieważ nie wszystkie narzędzia posiadają otwarte interfejsy i czasami wykorzystują własne niestandardowe języki skryptowe. Rozsądnie jest używać narzędzi posiadających wtyczki do otwartych platform lub API. To zagwarantuje lepszą możliwość wykorzystania skryptów do testów w przyszłości.

Dla każdego typu narzędzia, niezależnie od fazy, w której ma być użyte, należy rozważyć niżej wymienione cechy. Cechy te można wykorzystać zarówno przy ocenie narzędzi, jak i podczas tworzenia własnych. W każdym z tych obszarów narzędzie może być mocne lub słabe. Poniższa lista jest użyteczna przy porównywaniu podobnych narzędzi.

- Analiza (zrozumienie koncepcji, wejścia, informacje dostarczane ręcznie lub automatycznie)
- Projektowanie (ręczne, generowanie automatyczne)
- Wybór (ręczny, przypadki testowe wybierane automatycznie według różnych kryteriów np. pokrycia)

- Wykonanie (ręczne, automatyczne, sterujące, restartujące itp.)
- Ocena (np. wyrocznia testowa) i prezentacja - nazywane też często logowaniem i raportowaniem (ręczne, automatyczne np. porównawcze, według formy, standardu, wygenerowane według kryteriów)

9.2.7 Użycie narzędzi otwartych „Open Source”

Narzędzia używane do testów systemów krytycznych ze względu na bezpieczeństwo muszą posiadać certyfikaty zgodności z odpowiednimi standardami. Używanie narzędzi *open source* do testowania systemów krytycznych ze względu na bezpieczeństwo nie jest rekomendowane, chyba że posiadają odpowiednie certyfikaty.

Jakość narzędzi *open source* zależy od ilości użytkowników, historii oraz użytkowania i nie należy zakładać, że są one bardziej (ani mniej) dokładne od narzędzi komercyjnych.

Dla każdego narzędzia testowego należy wykonać ocenę jakości, aby poznać jego dokładność. W przypadku niektórych typów narzędzi łatwiej jest pomylić pozytywną ocenę wyników testów z błędem w wykonaniu testów (np. narzędzie pominęło wykonanie testu i nie zaraportowało, co zostało pominięte). Należy starannie rozważyć opłaty licencyjne. Może też istnieć oczekiwanie, że modyfikacje kodu wykonane w celu ulepszenia narzędzia zostaną udostępnione publicznie.

9.2.8 Rozwój własnych narzędzi

Wiele narzędzi testowych powstaje, by zaspokoić potrzeby pojedynczego testera lub projektanta i przyspieszyć jego pracę. Innymi powodami rozwoju własnych narzędzi testowych może być brak odpowiednich narzędzi komercyjnych lub wykorzystywanie własnego sprzętu czy środowiska testowego. Narzędzie takie często efektywnie wykonuje zadania, do których zostało stworzone, ale też często jest silnie zależne od osoby twórcy. Powinno ono zostać udokumentowane tak, aby mogło być utrzymywane przez innych. Ważne jest również rozważenie celowości, korzyści oraz możliwych problemów przed rozszerzeniem jego użycia w organizacji. Dla takich narzędzi często pojawiają się nowe wymagania i są rozwijane daleko poza początkowy sposób użycia, co może nie być korzystne.

9.2.9 Klasyfikacja narzędzi testowych

Oprócz podziału narzędzi według zadań, które wspierają (taki podział wykorzystany jest w syllabusie poziomu podstawowego), istnieją inne sposoby klasyfikacji narzędzi testowych:

- według poziomu testów (modułowe, integracyjne, systemowe, akceptacyjne)
- według rodzaju awarii, które rozpoznają i wspierają
- według podejścia do testowania lub techniki testowania (patrz dyskusja poniżej)
- według celów (pomiary, sterowniki testowe, logowanie, porównywanie)
- według dziedziny (symulacja sygnalizacji i ruchu, sieci, protokoły, transakcje, ekrany TV, systemy eksperckie)
- według wspieranego obszaru testowania (wprowadzanie danych, środowisko, konfiguracja i inne)
- według sposobu wdrożenia („z półki”, platformy do adaptacji, wtyczki, standardowe lub certyfikacyjne zestawy testów, wytworzone lokalnie)

I na koniec, narzędzia mogą być pogrupowane według ich użytkowników, takich jak kierownicy testów, analitycy testów i techniczni analitycy testów. W dalszych sekcjach tego rozdziału używane jest to grupowanie odzwieciedlajace moduły niniejszego syllabusa. Syllabus poziomu podstawowego zawiera rozdział opisujący narzędzia testowe. Poniższe sekcje zawierają dodatkowe aspekty tych narzędzi.

9.3 Kategorie narzędzi testowych

Sekcja ta ma następujące cele:

- dostarczenie dodatkowych informacji na temat kategorii narzędzi już omówionych w syllabusie poziomu podstawowego w rozdziale 6, takich jak narzędzia do zarządzania testami, narzędzia do wykonywania testów oraz narzędzia do testów wydajnościowych
- wprowadzenie nowych kategorii narzędzi

W celu uzyskania ogólnych informacji na temat kategorii narzędzi nie omawianych w tym podrozdziale należy sięgnąć do rozdziału 6 syllabusu poziomu podstawowego.

9.3.1 Narzędzia do zarządzania testami

Ogólne informacje na temat narzędzi do zarządzania testami znajdują się w podrozdziale 6.1.2 w syllabusie poziomu podstawowego.

Narzędzia do zarządzania testami powinny pozwalać na śledzenie następujących informacji:

- powiązania pomiędzy artefaktami testowymi
- dane na temat skomplikowanych środowisk
- dane o równoległym wykonaniu zestawów testów na różnych środowiskach w tej samej sesji testowej w różnych miejscach testowania (organizacjach testowych)
- metryki:
  - warunki testowe
  - przypadki testowe
  - czas wykonania (np. przypadku testowego, zestawu testów, testów regresywnych) i innych ważnych czasów, włącznie ze średnimi, które mogą wpływać na decyzje kierownicze
  - liczba przypadków testowych, artefaktów testowych i środowisk testowych
  - wskaźniki zaliczenia testów
  - liczba oczekujących testów (i powody, dla których nie zostały jeszcze wykonane)
  - trendy
  - wymagania
  - powiązania i możliwości śledzenia powiązań pomiędzy artefaktami testowymi
- pojęcia używane w narzędziach do zarządzania testami:
  - organizator artefaktów testowych, repozytorium oraz sterownik przypadków testowych
  - warunki testowe i środowiska testowe

- zestawy regresywne, sesje testowe
- logowanie i informacja o obsłudze awarii
- restart i reincjalizacja środowiska
- metryki związane z artefaktami testowymi (dokumentacją testową), aby udokumentować postęp testów

Narzędzia do zarządzania testami używane są przez kierowników testów, analityków testów oraz technicznych analityków testów.

9.3.2 Narzędzia do wykonywania testów

Ogólne informacje na temat narzędzi do wykonywania testów zawiera rozdział 6.1.5 syllabusa poziomu podstawowego.

Narzędzia do wykonywania testów są najczęściej używane w celu uruchamiania testów oraz sprawdzenia ich wyników przez analityków testów i technicznych analityków testów na wszystkich poziomach testowania. Celem użycia narzędzia do wykonywania testów jest zwykle jedno lub więcej z poniższych:

- redukcja kosztów (pracochłonności lub czasu)
- wykonanie większej ilości testów
- zwiększenie powtarzalności testów

Narzędzia do wykonywania testów są używane najczęściej do automatyzacji testów regresywnych.

Narzędzia do wykonywania testów działają na zasadzie wykonywania zbioru instrukcji języka programowania, często nazywanego językiem skryptowym. Instrukcje są wydawane narzędziu na bardzo szczegółowym poziomie, który specyfikuje poszczególne naciśnięcia przycisków, klawiszy oraz ruchy myszy. Powoduje to, że skrypty testowe są mało odporne na zmiany w testowanym oprogramowaniu, a w szczególności na zmiany GUI.

Bazą do tworzenia skryptu może być nagranie (z wykorzystaniem narzędzia rejestrująco-odtwarzającego) lub napisany od początku lub zmieniony skrypt z wykorzystaniem istniejących skryptów, wzorców lub słów kluczowych. Skrypt jest programem i działa tak, jak każde inne oprogramowanie. Rejestrowanie (lub nagrywanie) może zostać wykorzystane również do zapisania śladu po testach niesystematycznych. Większość narzędzi do wykonywania testów posiada komparatory, umożliwiające porównania wyników rzeczywistych z zapisanymi wynikami oczekiwanymi. W testowaniu (tak jak w programowaniu) istnieje tendencja do odchodzenia od szczegółowych niskopoziomowych instrukcji na rzecz wysokopoziomowych języków, z wykorzystaniem bibliotek, makr oraz podprogramów. W testowaniu opartym na słowach kluczowych lub słowach akcji ciąg instrukcji otrzymuje jedną etykietę. Główną korzyścią jest tutaj oddzielenie instrukcji od danych. Jest to analogiczny pomysł do używania wzorców w skryptach w celu zminimalizowania pracy do wykonania przez użytkownika.

Głównymi powodami, dla których niektóre narzędzia nie zdają egzaminu, są małe umiejętności programistyczne oraz brak zrozumienia, że narzędzia testowe rozwiązują tylko część problemów automatyzacji wykonywania testów. Należy pamiętać, że automatyzacja wykonania testów wymaga zarządzania, wysiłku, umiejętności i uwagi np. odpowiedniej architektury testowej i zarządzania konfiguracją. Oznacza to też, że skrypty testowe również mogą mieć usterki. Wykorzystanie architektury testowej może uniezależnić od konkretnego dostawcy narzędzi. Kiedy narzędzie zostaje nabyte, panuje tendencja do myślenia, że trzeba przestrzegać standardów narzędzia, na przykład w strukturze i nazewnictwie skryptów. Automatyzując przygotowanie do wykonania testów można jednak zbudować interfejs pomiędzy własnym sposobem organizacji testów a umiejscowieniem testów do uruchomienia przez narzędzie.

9.3.3 Narzędzia do debagowania i rozwiązywania problemów

Rozwiązywanie problemów może wymagać wykorzystania narzędzi w celu namierzenia obszaru występowania defektu. Może być również potrzebne w przypadku systemu, gdzie nie do końca jest wiadome, jaka usterka spowodowała zauważoną awarię. Narzędziami do rozwiązywania problemów są narzędzia do śledzenia oraz symulatory środowisk wykorzystywane do interakcji z oprogramowaniem lub do wydobywania informacji z systemu w celu określenia lokalizacji usterki.

Przy użyciu narzędzi do debagowania i śledzenia programiści reprodukują usterki i badają stan programów. Debagery pozwalają programistom na:

- wykonanie programu linia po linii
- zatrzymanie programu na dowolnym wyrażeniu
- ustawianie i sprawdzanie wartości zmiennych

Powinno być jasno powiedziane, że debagowanie (i debager) jest powiązane z testowaniem, ale testowaniem (ani narzędziem testowym) nie jest. Narzędzia do śledzenia i debagowania mogą być używane przez testerów do rozwiązywania problemów, do sprawniejszej lokalizacji przyczyn defektów i do ustalenia, gdzie wysłać raport o usterce. Narzędzia do debagowania, śledzenia i rozwiązywania problemów są używane głównie przez technicznych analityków testów.

9.3.4 Narzędzia do posiewu usterek i wstrzykiwania błędów

Posiew usterek i wstrzykiwanie błędów są dwiema różnymi technikami, które mogą być używane w testowaniu. W posiewie usterek używane jest narzędzie podobne do kompilatora, które w usystematyzowany sposób tworzy pojedynczą lub ograniczoną liczbę typów usterek w kodzie. Narzędzia takie często używane są w połączeniu z techniką testowania różnicowego (testowanie mutacji) i są czasami nazywane narzędziami do testowania mutacji.

Wstrzykiwanie błędów jest nakierowane na zmianę interfejsów do modułów testowych (kiedy kod źródłowy nie jest dostępny), ale mogłoby również specjalnie wstrzykiwać konkretny błąd, aby sprawdzić, czy oprogramowanie sobie z nim poradzi (tolerowanie usterek) lub czy test wzięty z zestawu testów wykrywa specjalnie wstrzykniętą usterkę. Narzędzia do posiewu usterek i wstrzykiwania błędów są wykorzystywane głównie na poziomie kodu przez technicznych analityków

testów, ale również możliwa jest zmiana danych w bazie danych przez analityka testów lub wstrzykiwanie błędów do strumienia danych w celu przetestowania zachowania systemu.

9.3.5 Symulatory i emulatory

Symulatory wspierają testy, kiedy kod lub inne systemy są niedostępne, drogie lub ich użycie jest niepraktyczne (np. przy testowaniu oprogramowania działającego w przypadku wycieku z elektrowni atomowych). Niektóre symulatory i narzędzia jarzm testowych potrafią również udawać błędne zachowanie systemu, można więc sprawdzić błędne stany i błędne scenariusze. Głównym ryzykiem związanym z wykorzystaniem tych narzędzi jest możliwość niewykrycia błędów związanych z zasobami (takich jak chronologia zdarzeń), co jest bardzo ważne dla niektórych typów systemów.

Emulatory są szczególną kategorią symulatorów, do której należy oprogramowanie napisane po to, żeby naśladować sprzęt. Korzyścią z używania emulatorów jest możliwość wykonania bardziej skomplikowanych testów. Inną korzyścią płynącą z używania emulatorów jest możliwość śledzenia, debagowania, powtórzenia problemów związanych z czasem, co może być niemożliwe w rzeczywistym systemie. Stworzenie emulatorów jest kosztowne, ale korzyści z analizy, kiedy system jest uruchamiany w zwolnionym trybie, dla niektórych systemów równoległych, zależnych od czasu i złożonych są nieocenione.

W zależności od wymaganego rodzaju emulacji, narzędzia te są używane przez analityków testów lub technicznych analityków testów.

9.3.6 Narzędzia do analizy statycznej i dynamicznej

W celu uzyskania ogólnych informacji na temat testowania statycznego oraz narzędzi do analizy dynamicznej należy sięgnąć do syllabusa poziomu podstawowego do rozdziałów 3.3 Analiza statyczna przy pomocy narzędzi, 6.1.3 Testowanie statyczne wspierane narzędziami oraz 6.1.6 Testowanie wydajności i monitorowanie wspierane narzędziami.

9.3.6.1 Narzędzia do analizy statycznej

Narzędzia do analizy statycznej mogą być używane w dowolnym momencie cyklu życia oprogramowania i w dowolnej fazie jego rozwoju, w zależności od rodzaju pomiarów przez nie wykonywanych.

Narzędzia do analizy statycznej raportują swoje wyniki w postaci ostrzeżeń. Ostrzeżenia, które są bezpodstawne nazywane są wynikami fałszywie pozytywnymi. Wynik (prawdziwie) pozytywny to rzeczywiste usterki, które mogą prowadzić do awarii w czasie wykonania programu. Rozróżnienie wyników fałszywie i prawdziwie pozytywnych może być trudne i czasochłonne, ponieważ wymaga poprawnego rozpoznania problemu. Nowsze narzędzia do analizy statycznej potrafią korzystać z informacji o dynamicznym wiązaniu w czasie kompilacji, są skuteczniejsze w znajdowaniu prawdziwych usterek i dają mniejszą ilość wyników fałszywie pozytywnych. Narzędzi tych używają techniczni analitycy testów.

9.3.6.2 Narzędzia do analizy dynamicznej

Narzędzia do analizy dynamicznej dostarczają informacji na temat stanu oprogramowania w czasie jego działania. Narzędzia te są najczęściej używane do wyszukiwania niezainicjowanych wskaźników, sprawdzania arytmetyki wskaźników, monitorowania alokacji, użycia oraz dealokacji pamięci w celu wykrycia wycieków pamięci i pokazania innych błędów trudnych do wykrycia statycznie. Narzędzia do diagnostyki pamięci powinny być używane na wielu poziomach podczas testów dużych i złożonych systemów, ponieważ problemy z pamięcią pojawiają się dynamicznie. Należy zauważyć, że różne komercyjne narzędzia mogą być różnie zaimplementowane i dlatego mogą raportować różne typy problemów związanych z pamięcią lub zasobami (stos, sterta). Wniosek z tego jest taki, że różne narzędzia mogą wykrywać różne problemy. Narzędzia diagnozujące pamięć są szczególnie użyteczne przy niektórych językach programowania (C, C++), w których zarządzanie pamięcią zostało pozostawione programistom. Narzędzi tych używają techniczni analitycy testów.

9.3.7 Automatyzacja oparta o słowa kluczowe

Słowa kluczowe (czasami nazywane też „słowami akcji") są używane głównie (chociaż nie tylko) do reprezentowania wysokopoziomowych biznesowych interakcji z systemem (np. „anuluj zamówienie"). Każde słowo kluczowe zwykle reprezentuje kilka bardziej szczegółowych interakcji z testowanym systemem. Ciągi słów kluczowych (zwierające odpowiednie dane testowe) służą do specyfikowania przypadków testowych. [Buwalda01]

Podczas automatyzacji testów słowo kluczowe zostaje zaimplementowane jako jeden lub więcej wykonywalnych skryptów testowych. Narzędzia czytają przypadki testowe napisane przy użyciu słów kluczowych i wywołują odpowiednie skrypty testowe, które je implementują. Skrypty są pisane w bardzo modułowy sposób, żeby ułatwić mapowanie na poszczególne słowa kluczowe. Do implementacji tych modułowych skryptów konieczne jest posiadanie umiejętności programowania.

Główne korzyści z automatyzacji opartej na słowach kluczowych to:

- Słowa kluczowe mogą być definiowane przez ekspertów od konkretnej aplikacji lub dziedziny biznesowej. Może to spowodować, że specyfikacja przypadków testowych będzie bardziej efektywna.
- Osoba posiadająca głównie wiedzę dziedzinową może skorzystać z automatycznego wykonania przypadków testowych (wtedy, gdy słowa kluczowe zostały zaimplementowane jako skrypty)
- Przypadki testowe zaimplementowane przy użyciu słów kluczowych są łatwiejsze w utrzymaniu ponieważ zmniejsza się prawdopodobieństwo konieczności ich modyfikacji przy mniejszych zmianach testowanego oprogramowania.
- Specyfikacja przypadków testowych jest oderwana od ich implementacji. Słowa kluczowe mogą zostać zaimplementowane przy użyciu różnych języków skryptowych i narzędzi.

Automatyzacja oparta na słowach kluczowych jest używana głównie przez ekspertów dziedzinowych oraz analityków testowych.

9.3.8 Narzędzia do testów wydajnościowych

Ogólne informacje na temat narzędzi do testów wydajnościowych zostały zamieszczone w syllabusie poziomu podstawowego w rozdziale 6.1.6 Testowanie wydajności i monitorowanie wspierane narzędziami.

Narzędzia do testów wydajnościowych spełniają dwie główne funkcje:

- generowanie obciążenia
- pomiar i analiza odpowiedzi systemu na zadane obciążenie

Generowanie obciążenia jest wykonywane przez zaimplementowanie zdefiniowanego wcześniej profilu operacyjnego (patrz sekcja 5.3.3) jako skryptu. Skrypt może zostać nagrany wcześniej dla jednego użytkownika (jeżeli to możliwe, to przy użyciu narzędzia rejestrująco-odtwarzającego), a później zaimplementowany dla celów konkretnego profilu operacyjnego przy użyciu narzędzia do testów wydajnościowych. Ta implementacja musi brać pod uwagę różnicowanie danych pomiędzy transakcjami (lub zbiorami transakcji).

Narzędzia do testów wydajnościowych generują obciążenie przez symulowanie dużej liczby użytkowników (wirtualnych) z konkretnymi rozmiarami danych wejściowych. Porównując ich działanie z narzędziami rejestrująco-odtwarzającymi, wiele narzędzi do testów wydajnościowych odtwarza interakcje z systemem na poziomie protokołu komunikacyjnego, a nie przez symulację interakcji użytkownika poprzez graficzny interfejs użytkownika. Tylko ograniczona liczba narzędzi do generowania obciążenia potrafi generować obciążenie sterując aplikacją poprzez jej interfejs użytkownika.

Narzędzia do testów wydajnościowych wykonują szeroki wachlarz pomiarów, w celu umożliwienia analizy w trakcie i po wykonaniu testów. Typowe metryki i raporty to:

- liczba symulowanych użytkowników
- liczba i typ transakcji generowanych przez symulowanych użytkowników
- czasy odpowiedzi na poszczególne transakcje wykonywane przez użytkowników
- raporty bazujące na logach testowych i grafy pokazujące czasy odpowiedzi dla różnych obciążeń

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

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

Narzędzia do testów wydajnościowych zwykle są nabywane z powodu pracochłonności wymaganej do ich wytworzenia. Jednakże może być zasadnym rozwijanie konkretnego narzędzia, jeżeli

ograniczenia techniczne uniemożliwiają użycie gotowych produktów lub gdy profil obciążenia i funkcje do zaimplementowania są relatywnie proste. Narzędzia do testów wydajnościowych zwykle są używane przez technicznych analityków testów.

Uwaga: Usterki związane z wydajnością często mają duży wpływ na testowane oprogramowanie. Gdy spełnienie wymagań na wydajność jest bardzo istotne, często użyteczne okazuje się przeprowadzenie testów wydajnościowych krytycznych modułów (przez zaślepki i sterowniki testowe) bez czekania na testy systemowe.

9.3.9 Narzędzia webowe

Narzędzia testujące hiperłącza wykorzystywane są do skanowania i sprawdzania, czy na danej witrynie nie występują nieważne lub brakujące łącza. Niektóre narzędzia dostarczają też dodatkowych informacji, takich jak graf architektury (drzewo witryny), prędkość ściągania i rozmiar ściągniętych danych (na URL), trafienia oraz wolumeny. Narzędzia te mogą być również pomocne w monitorowaniu wypełniania umowy utrzymania i systematycznego poprawiania (ang. *Service Level Agreement - SLA*). Narzędzi tych używają analitycy testów oraz techniczni analitycy testów.

10. Umiejętności interpersonalne – skład zespołu

Terminologia

Niezależność testowania

10.1 Wprowadzenie

Wszyscy wykwalifikowani testerzy powinni być świadomi istnienia indywidualnych cech wymaganych, aby dobrze wykonywać konkretne zadania. Rozdział ten w pierwszej kolejności skupia się na tych indywidualnych umiejętnościach, po czym omawia szereg zagadnień specyficznych dla kierowników testów, takich jak dynamika zespołu, jego organizacja, motywacja i komunikacja.

10.2 Umiejętności indywidualne

Zdolność danej osoby do testowania oprogramowania może być uzyskana poprzez doświadczenie lub szkolenie w różnych obszarach pracy. Każdy z poniższych aspektów może przyczynić się do poszerzenia wiedzy testera:

- Posługiwanie się systemami oprogramowania
- Znajomość dziedziny lub biznesu
- Udział w różnych fazach procesu wytwarzania oprogramowania, włączając w to analizę, rozwój i pomoc techniczną
- Udział w testowaniu oprogramowania

Użytkownicy systemów dobrze znają je od strony użytkownika i mają szeroką wiedzę na temat tego, jak system działa, jakie awarie i w których częściach systemu mogą mieć największy negatywny wpływ i jaka powinna być oczekiwana reakcja systemu. Użytkownicy posiadający wiedzę dziedzinową wiedzą, które obszary są najważniejsze w danym biznesie i jak one wpływają na zdolność systemu do spełnienia postawionych przed nim wymagań. Ta wiedza może zostać wykorzystana, aby pomóc określić priorytety dla czynności testowych, stworzyć realistyczne dane testowe i przypadki testowe, oraz weryfikować lub dostarczać przypadki użycia.

Wiedza o procesie wytwarzania oprogramowania (analiza wymagań, projektowanie i kodowanie) daje wgląd w sposób, w jaki mogą zostać popełnione błędy, gdzie mogą zostać wykryte i jak im zapobiegać. Doświadczenie w dziedzinie pomocy technicznej daje wiedzę o doświadczeniach użytkowników, ich oczekiwaniach i wymaganiach dotyczących użyteczności. Doświadczenie w rozwijaniu oprogramowania jest ważne podczas wykorzystania specjalistycznych narzędzi do automatyzacji testów, których użycie wymaga umiejętności programistycznych i projektowych.

Specyficzne umiejętności testowania oprogramowania obejmują: umiejętność analizowania specyfikacji, uczestniczenia w analizie ryzyka, projektowania przypadków testowych, oraz staranność w przeprowadzaniu testów i rejestrowaniu wyników.

Szczególnie ważne dla kierownika testów jest posiadanie wiedzy, umiejętności i doświadczenia w zarządzaniu projektami, gdyż zarządzanie testami jest podobne do realizacji projektu, np. tworzenie planu, śledzenie postępów i raportowanie do interesariuszy.

Umiejętności interpersonalne, takie jak udzielanie i przyjmowanie krytyki, wywieranie wpływu i negocjowanie są ważne w roli związanej z testowaniem. Tester kompetentny pod względem

technicznym może się nie sprawdzać w swojej roli, jeśli nie posiada i nie wprowadza w życie wymaganych umiejętności interpersonalnych. W dodatku do efektywnej pracy z innymi, profesjonalista z zakresu testowania musi być również osobą zorganizowaną, skrupulatną i posiadać dobre umiejętności komunikacyjne (zarówno w kontekście komunikacji pisanej, jak i werbalnej).

10.3 Dynamika zespołu testowego

Selekcja personelu jest jedną z najważniejszych funkcji ról kierowniczych w organizacji. Poza specyficznymi cechami indywidualnymi jest wiele elementów wymaganych w danej pracy, które muszą być wzięte pod uwagę. Podczas wyboru pracownika, który ma dołączyć do zespołu, należy wziąć pod uwagę całą jego dynamikę. Czy ta osoba będzie pasowała do zespołu ze względu na umiejętności i typ osobowości? Ważne jest, aby rozważyć korzyści płynące z posiadania w zespole testowym różnorodnych typów osobowości oraz osób o różnych umiejętnościach technicznych. Silny zespół testowy jest w stanie poradzić sobie z wieloma projektami o różnej złożoności, ale i z powodzeniem radzić sobie w interakcjach z innymi członkami zespołu projektowego.

Nowi członkowie zespołu muszą być szybko integrowani z zespołem i poddani odpowiedniemu nadzorowi. Każda osoba powinna mieć ściśle zdefiniowaną rolę w zespole. Może ona być oparta na wynikach indywidualnej oceny. Celem jest to, aby każdy odnosił sukcesy osobiste, jednocześnie wnosząc wkład w sukces całego zespołu. Dokonuje się tego głównie poprzez dopasowanie typów osobowości do ról zespołowych i rozwijanie wrodzonych umiejętności jednostki, jak i poprzez rozszerzanie zestawu jej umiejętności.

Należy pamiętać, że idealna osoba jest rzadko dostępna, ale można zbudować silny zespół równoważąc słabe strony poszczególnych osób mocnymi stronami innych. Wzajemne szkolenie współpracowników wewnątrz zespołu jest konieczne, aby pielęgnować i budować wiedzę zespołu, a także zwiększać jego elastyczność.

10.4 Dopasowanie testowania do organizacji

Organizacje bardzo się różnią w sposobie, w jaki testowanie jest wpasowane w ich strukturę. Mimo że jakość jest odpowiedzialnością każdej osoby przez cały cykl życia oprogramowania, niezależny zespół testowy może przyczynić się do stworzenia produktu odpowiedniej jakości. Poziom niezależności testowania może się w praktyce znacznie różnić, jak widać na poniżej liście, ułożonej w kolejności od najmniejszego do największego stopnia niezależności:

- Brak niezależnych testerów
  - W tym przypadku nie ma niezależności i programista testuje swój własny kod
  - Programista, jeśli będzie miał czas na przetestowanie, będzie decydował, czy kod działa tak, jak on planował, co może lub nie pasować do rzeczywistych wymagań
  - Programista może szybko naprawić jakiekolwiek znalezione defekty
- Testowanie jest wykonywane przez innego programistę, niż ten, który napisał kod
  - Niewielki stopień niezależności między programistą a testerem
  - Programista testujący kod innego developera może mieć opory przy zgłaszaniu defektów
  - Uwaga programisty testującego kod jest zazwyczaj skupiona na pozytywnych przypadkach testowych
- Testowanie jest dokonywane przez testera (lub zespół testowy) będącego częścią zespołu rozwojowego

- Tester (lub zespół testowy) będzie raportował do kierownictwa projektu
- Uwaga testera jest bardziej skupiona na weryfikowaniu zgodności z wymaganiami
- Ponieważ tester jest członkiem zespołu rozwojowego, może mieć obowiązki programistyczne w dodatku do testowania

- Testerzy z organizacji biznesowej, wspólnoty użytkowników, lub innych technicznych organizacji niemających związku z rozwojem oprogramowania
  - Niezależne raportowanie do interesariuszy
  - Jakość znajduje się w centrum uwagi zespołu
  - Rozwój umiejętności i szkolenia są ukierunkowane na testowanie
- Zewnętrzni specjaliści testowi wykonujący testowanie dla specyficznych celów testowych
  - Celami testowymi mogą być użyteczność, bezpieczeństwo lub wydajność
  - Jakość powinna być w centrum uwagi tych specjalistów, jednak może to być zależne od struktury raportowania
- Testowanie jest wykonywane przez organizację zewnętrzną
  - Osiągnięta jest maksymalna niezależność
  - Transfer wiedzy i informacji może być niewystarczający
  - Potrzebne są jasno określone wymagania oraz dobrze zdefiniowana struktura komunikacji
  - Jakość zewnętrznej organizacji musi być regularnie audytowana

Są różne stopnie niezależności pomiędzy organizacjami zajmującymi się testowaniem a tymi, które specjalizują się w rozwijaniu oprogramowania. Ważne jest, aby zrozumieć, że może wystąpić sytuacja kompromisowa, w której większa niezależność skutkuje większą izolacją, a mniejszym przepływem wiedzy i informacji. Mniejszy poziom niezależności może podnieść poziom wiedzy, ale i prowadzić do konfliktu interesów. Poziom niezależności jest określony przez używany model wytwarzania oprogramowania, np. w metodykach zwinnych testerzy są zazwyczaj częścią zespołu rozwojowego.

Wszystkie z powyższych opcji mogą zostać w organizacji połączone. Testowanie może być wykonane zarówno wewnątrz organizacji rozwijającej oprogramowanie, jak i przez niezależną organizację testującą, a końcowa certyfikacja wykonana zostanie przez organizację zewnętrzną. Ważne jest, aby zrozumieć odpowiedzialności i oczekiwania w stosunku do każdego z etapów testowania i wyznaczyć te wymagania tak, by zmaksymalizować jakość końcowego produktu przy zachowaniu ograniczeń harmonogramowych i budżetowych.

Outsourcing, czyli zlecanie zadań osobom spoza firmy, jest jedną z form korzystania z zewnętrznych organizacji. Wynajęty zespół może być inną firmą, która dostarcza usługi testowe, może znajdować się na terenie firmy, lub poza nią, ale wewnątrz twojego kraju, lub w innym kraju (co bywa określane jako „off-shoring"). Outsourcing stawia wyzwania, szczególnie, jeśli zleceniobiorca znajduje się poza granicami kraju. Niektóre z kwestii, na które należy zwrócić uwagę, to:

- Różnice kulturowe
- Nadzór nad wynajętymi zasobami
- Transfer informacji, komunikacja
- Ochrona własności intelektualnej
- Zestawy umiejętności, rozwój umiejętności, oraz szkolenia
- Fluktuacje kadrowe
- Dokładne oszacowanie kosztów
- Jakość

10.5 Motywacja

Jest wiele sposobów motywowania osoby na stanowisku testera. Należą do nich:

- Uznanie dla wykonanej pracy
- Aprobata kadry kierowniczej
- Szacunek w zespole projektowym i wśród współpracowników
- Odpowiednie nagrody za dokonaną pracę (w tym wynagrodzenie za pracę, nagrody i premie)

Istnieją czynniki projektowe, które mogą sprawić, że te narzędzia motywacyjne staną się trudne w zastosowaniu. Na przykład tester może bardzo ciężko pracować nad projektem, którego termin realizacji jest niemożliwy do dotrzymania. Tester może robić wszystko, aby kierować uwagę zespołu na jakość, wkładać dodatkowy wysiłek i pracować w nadgodzinach, ale produkt może wciąż trafić na rynek wcześniej, niż powinien z powodu zewnętrznych wpływów. W wyniku tego produkt może być niskiej jakości, pomimo największego wysiłku testera. To może z kolei łatwo zniechęcić testera, jeśli jego wkład nie jest zrozumiany lub doceniony bez względu na to, czy produkt końcowy odniósł sukces.

Zespół testowy musi upewnić się, że śledzi odpowiednie metryki, aby udowodnić, że praca została dobrze wykonana, włączając w to testowanie, łagodzenie ryzyka i odpowiednie raportowanie wyników. Jeśli dane te nie zostaną zebrane i udostępnione, zespół może łatwo stracić motywację, kiedy nie otrzymuje aprobaty, której się spodziewa za dobrze wykonaną pracę.

Aprobata jest określana nie tylko poprzez tak nieuchwytne korzyści, jak szacunek i uznanie, jest ona również widoczna w możliwości awansu, skali wynagrodzenia i ścieżkach kariery. Jeśli grupa testowa nie jest szanowana, możliwości te mogą nie być dostępne.

Aprobata i uznanie są zdobywane, gdy jest oczywiste, że tester przyczynił się do wzrostu wartości projektu. W pojedynczym projekcie jest to osiągane najszybciej poprzez zaangażowanie testera w prace nad koncepcją oraz podtrzymanie tego zaangażowania przez cały projekt. Z biegiem czasu tester zyskuje uznanie i szacunek poprzez wkład w pozytywny rozwój projektu, ale ten wkład powinien być mierzony także pod względem redukcji kosztów uzyskania odpowiedniego poziomu jakości oraz zmniejszenia poziomu ryzyka.

10.6 Komunikacja

Komunikacja zespołowa występuje przede wszystkim na trzech poziomach:

- Dokumentacja produktów testowych: strategia testów, plan testów, przypadki testowe, sumaryczne raporty z testów, zgłoszenia defektów, itd.
- Informacja zwrotna na temat zrecenzowanych dokumentów: wymagania, specyfikacje funkcjonalne, przypadki użycia, dokumentacja testów modułowych, itd.
- Gromadzenie oraz rozpowszechnianie informacji: interakcje z programistami, z innymi członkami t zespołu testowego, z kierownictwem, itd.

Komunikacja musi być prowadzona w sposób profesjonalny, obiektywny i efektywny w celu budowy i utrzymania szacunku i uznania dla zespołu testowego. Podczas udzielania informacji zwrotnej, a szczególnie konstruktywnej informacji zwrotnej, na temat wyniku pracy innych ludzi, wymagana jest dyplomacja i obiektywność.

Komunikacja powinna być skupiona na osiągnięciu celów testowania i na poprawie jakości, zarówno produktów, jak i procesów używanych do tworzenia systemów oprogramowania. Testerzy komunikują się z wieloma odbiorcami, w tym użytkownikami, członkami zespołów projektowych, kierownictwem, zewnętrznymi zespołami testowymi i klientami. Komunikacja musi być efektywna dla odbiorców docelowych. Na przykład, raport trendu defektów stworzony dla zespołu programistów może być zbyt szczegółowy, by odpowiadał potrzebom podsumowania dla kierownictwa.

11. Odnośniki

11.1 Standardy

W tym rozdziale wymieniono standardy, do których odwoływano się w tym syllabusie.

11.1.1 Podział pod względem rozdziałów

- Rozdział 2

  BS-7925-2, IEEE 829, DO-178B/ED-12B

- Rozdział 3

  IEEE 829, DO-178B/ED-12B

- Rozdział 4

  BS 7925-2

- Rozdział 5

  ISO 9126

- Rozdział 6

  IEEE 1028

- Rozdział 7

  IEEE 829, IEEE 1044, IEEE 1044.1

11.1.2 Podział alfabetyczny

- [BS-7925-2] BS 7925-2 (1998) Software Component Testing

  Rozdział 2 i 4

- [IEEE 829] IEEE Std 829™ (1998/2005) IEEE Standard for Software Test Documentation (obecnie w przeglądzie)

  Rozdział 2 i 3

- [IEEE 1028] IEEE Std 1028™ (1997) IEEE Standard for Software Reviews

  Rozdział 6

- [IEEE 1044] IEEE Std 1044™ (1993) IEEE Standard Classification for Software Anomalies

  Rozdział 7

- [ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality

  Rozdział 5

- [ISTQB] ISTQB Glossary of terms used in Software Testing, wersja 2.0, 2007
- [RTCA DO-178B/ED-12B]: Software Considerations in Airborne systems and Equipment certification, RTCA/EUROCAE ED12B. 1992.

  Rozdział 2 i 3

11.2 Książki

[Beizer95] Beizer Boris, "Black-box testing", John Wiley & Sons, 1995, ISBN 0-471-12094-4

[Black02]: Rex Black, "Managing the Testing Process (2nd edition)", John Wiley & Sons: New York, 2002, ISBN 0-471-22398-0

[Black03]: Rex Black, "Critical Testing Processes", Addison-Wesley, 2003, ISBN 0-201-74868-1

[Black07]: Rex Black, "Pragmatic Software Testing", John Wiley and Sons, 2007, ISBN 978-0-470-12790-2

[Burnstein03]: Ilene Burnstein, "Practical Software Testing", Springer, 2003, ISBN 0-387-95131-8

[Buwalda01]: Hans Buwalda, "Integrated Test Design and Automation" Addison-Wesley Longman, 2001, ISBN 0-201-73725-6

[Copeland03]: Lee Copeland, "A Practitioner's Guide to Software Test Design", Artech House, 2003, ISBN 1-58053-791-X

[Craig02]: Craig, Rick David; Jaskiel, Stefan P., "Systematic Software Testing", Artech House, 2002, ISBN 1-580-53508-9

[Gerrard02]: Paul Gerrard, Neil Thompson, "Risk-based e-business testing", Artech House, 2002, ISBN 1-580-53314-0

[Gilb93]: Gilb Tom, Graham Dorothy, "Software inspection", Addison-Wesley, 1993, ISBN 0-201-63181-4

[Graham07]: Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black "Foundations of Software Testing", Thomson Learning, 2007, ISBN 978-1-84480-355-2

[Grochmann94]: M. Grochmann (1994), Test case design using Classification Trees, in: conference proceeedings STAR 1994

[Jorgensen02]: Paul C.Jorgensen, "Software Testing, a Craftsman's Approach second edition", CRC press, 2002, ISBN 0-8493-0809-7

[Kaner02]: Cem Kaner, James Bach, Bret Pettichord; "Lessons Learned in Software Testing"; Wiley, 2002, ISBN: 0-471-08112-4

[Koomen99]: Tim Koomen, Martin Pol, "Test Process Improvement", Addison-Wesley, 1999, ISBN 0-201-59624-5.

[Myers79]: Glenford J.Myers, "The Art of Software Testing", John Wiley & Sons, 1979, ISBN 0-471-46912-2

[Pol02]: Martin Pol, Ruud Teunissen, Erik van Veenendaal, "Software Testing: A Guide to the Tmap Approach", Addison-Wesley, 2002, ISBN 0-201-74571-2

[Splaine01]: Steven Splaine, Stefan P.,Jaskiel, "The Web-Testing Handbook", STQE Publishing, 2001, ISBN 0-970-43630-0

[Stamatis95]: D.H. Stamatis, "Failure Mode and Effect Analysis", ASQC Quality Press, 1995, ISBN 0-873-89300

[vanVeenendaal02]: van Veenendaal Erik, "The Testing Practitioner", UTN Publsihing, 2002, ISBN 90-72194-65-9

[Whittaker03]: James Whittaker, "How to Break Software", Addison-Wesley, 2003, ISBN 0-201-79619-8

[Whittaker04]: James Whittaker and Herbert Thompson, "How to Break Software Security", Peason / Addison-Wesley, 2004, ISBN 0-321-19433-0

11.3 Inne odnośniki

Poniższe referencje odnoszą się do informacji dostępnej w Internecie.

Pomimo iż te odnośniki zostały sprawdzone przed publikacją syllabusa poziomu zaawansowanego, ISTQB nie odpowiada za materiały, które przestały być dostępne.

- Rozdział 5
  - www.testingstandards.co.uk
- Rozdział 6

- Taksonomia błędów: www.testingeducation.org/a/bsct2.pdf
- Przykładowa taksonomia błędów stworzona w oparciu o prace Borisa Beizera: inet.uni2.dk/~vinter/bugtaxst.doc
- Dobry przegląd różnych taksonomii: testingeducation.org/a/bugtax.pdf
- Heuristic Risk-Based Testing, James Bach, wywiad na stronie WhatIsTesting.com, www.whatistesting.com/interviews/jbach.htm
- www.satisfice.com/articles/et-article.pdf
- Z „Exploratory & Risk-Based Testing (2004) www.testingeducation.org"
- Exploring Exploratory Testing, Cem Kaner i Andy Tikam, 2003
- Pettichord, Bret, "An Exploratory Testing Workshop Report", www.testingcraft.com/exploratorypettichord

- Rozdział 9
  - www.sei.cmu.edu/cmmi/adoption/pdf/cmmi-overview06.pdf
  - TMMi www.tmmifoundation.org

12. Dodatek A – Podstawa syllabusa

Cele certyfikacji dla poziomu zaawansowanego:

- Uzyskanie uznania dla testowania jako istotnej zawodowej specjalizacji w inżynierii oprogramowania
- Dostarczenie standardowej ścieżki rozwoju kariery testera
- Umożliwienie zawodowo certyfikowanym testerom uzyskania uznania przez pracodawców, klientów oraz współpracowników oraz zwiększenie rozpoznawalności testerów
- Promowanie dobrych i spójnych praktyk w zakresie testowania we wszystkich dyscyplinach inżynierii oprogramowania
- Identyfikacja tematów związanych z testowaniem, które są istotne oraz stanowią wartość dla przemysłu
- Umożliwienie dostawcom oprogramowania zatrudnianie certyfikowanych testerów i uzyskanie przewagi konkurencyjnej przez reklamowanie swojej polityki zatrudniania
- Umożliwienie testerom i innym osobom zainteresowanym testowaniem uzyskania międzynarodowego certyfikatu z testowania

Kryteria wejściowe dla tego certyfikatu

Warunki wejściowe przystąpienia do egzaminu na Certyfikat Certyfikowany Tester – Poziom Zaawansowany:

- posiadanie certyfikatu poziomu podstawowego wystawionego przez komisję egzaminacyjną lub organizację narodową uznawaną przez ISTQB®
- posiadanie odpowiedniej liczby lat doświadczenia w testowaniu lub wytwarzaniu oprogramowania, jaka zostanie ustalona przez komisję egzaminacyjną lub organizację narodową, która będzie przeprowadzała certyfikację
- stosowanie się do kodeksu etycznego zawartego w syllabusie

Jest także zalecane, aby kandydaci ukończyli kurs, który został akredytowany przez jedną z organizacji członkowskich ISTQB (jednak nie jest to warunek konieczny, aby podejść do egzaminu).

Certyfikaty Practitioner lub Advanced z testowania oprogramowania (wystawione przez jednostkę akredytowaną przez ISTQB®) przyznane zanim ten międzynarodowy certyfikat został utworzony zostaną uznane za równoważne. Certyfikat poziomu zaawansowanego nie wygasa i nie musi być odnawiany. Data, kiedy został przyznany, jest widoczna na certyfikacie.

W każdym uczestniczącym kraju zasady lokalne są ustalane i kontrolowane przez organizację narodową[10]. Obowiązki organizacji narodowej są wyznaczane przez ISTQB® dla każdego kraju osobno.

[10] W Polsce jest nią SJSI [przypis SJSI].

Obowiązki zawierają akredytowanie centrów szkoleniowych i organizowanie egzaminów, wprost lub pośrednio przez jedną lub więcej komisji egzaminacyjnych, które mają umowę z ISTQB®.

13. Dodatek B – Informacja dla Czytającego

13.1 Komisje Egzaminacyjne

Syllabus poziomu zaawansowanego wymaga wiedzy z zakresu syllabusa poziomu podstawowego w wersji z 2005 roku na poziomie określonym w tym syllabusie.

Jednostki egzaminacyjne akredytowane przez ISTQB® mogę tworzyć pytania oparte na każdym z tematów wspomnianych w syllabusie.

Zalecane jest, aby pytania miały przyznawane różne ilości punktów w zależności do poziomu tematu, do którego się odnoszą. Na przykład, pytanie odnoszące się do poziomu wiedzy K1 powinno dawać mniej punktów, niż pytanie odnoszące się do poziomu wiedzy K3, a pytanie z zakresu K4 powinno być za jeszcze więcej punktów.

13.2 Kandydaci i Dostawcy Szkoleń

Aby otrzymać Certyfikat Certyfikowany Tester – Poziom Zaawansowany kandydat musi posiadać certyfikat poziomu podstawowego, spełniać wymagania komisji egzaminacyjnej, która go egzaminuje oraz posiadać wystarczające doświadczenie praktyczne, aby kwalifikować się do egzaminu. Chcąc sprawdzić, jakie dokładnie doświadczenie jest wymagane należy zwrócić się do odpowiedniej komisji egzaminacyjnej. ISTQB® zaleca minimum pięcioletnią praktykę w inżynierii oprogramowania jako wymóg do przystąpienia do egzaminu na poziomie zaawansowany lub 3 lata, jeśli kandydat posiada ukończone studia (ang. *baccalaureate*).

Osiągnięcie odpowiedniego poziomu profesjonalizmu, aby uzyskać poziom zaawansowany w testowaniu oprogramowania, wymaga więcej niż tylko znajomości zawartości tego syllabusa. Kandydaci i dostawcy szkoleń powinni spędzić więcej czasu na poszerzaniu swojej wiedzy niż wymieniono w syllabusie.

Syllabus posiada listę referencji do książek i standardów, które kandydaci i dostawcy szkoleń mogą przeczytać, aby zrozumieć wybrane tematy w szerszym stopniu.

14. Dodatek C – Informacja dla Dostawców Szkoleń

14.1 Podział na moduły

Ten syllabus zawiera materiały dla trzech modułów nazwanych odpowiednio:

- Kierownik Testów - Poziom Zaawansowany
- Analityk Testów - Poziom Zaawansowany
- Techniczny Analityk Testów - Poziom Zaawansowany

Zdanie wszystkich trzech modułów pozwala kandydatowi otrzymać certyfikat *"Full Advanced Level Testing Professional".*

14.2 Czas trwania szkoleń

14.2.1 Czas ze względu na moduł

Zalecany czas potrzebny na przekazanie wiedzy dla 3 różnych ról przedstawia się następująco:

- Kierownik Testów - Poziom Zaawansowany – 5 dni
- Analityk Testów - Poziom Zaawansowany – 5 dni
- Techniczny Analityk Testów - Poziom Zaawansowany – 5 dni

Czas ten jest oparty na ilości rozdziałów i zawartego w nich materiału do nauki dla poszczególnych modułów. Dokładny minimalny czas jest wypisany przy każdym rozdziale dla każdej z ról.

Dostawcy szkoleń mogą przeznaczyć na nauczanie więcej czasu niż jest zalecane tak jak kandydaci mogą spędzić więcej czasu czytając i badając wybrane zagadnienia. Program szkolenia nie musi pokrywać się z kolejnością zagadnień z syllabusa.

Szkolenia nie muszą się odbywać w 5 kolejnych dni. Dostawcy szkoleń mogą zorganizować swoje szkolenie w inny sposób, na przykład 3 + 2 dni na Zarządzanie Testami, lub 2 wspólne dni dla wszystkich a następnie 3 dni dla Analityków Testów i Technicznych Analityków Testów.

14.2.2 Części wspólne

Dostawcy szkoleń mogą uczyć wspólnych tematów tylko raz, aby zmniejszyć sumaryczny czas szkoleń i uniknąć powtórzeń, ale należy mieć na uwadze, że niektóre wspólne tematy wymagają spojrzenia z różnych perspektyw w zależności od modułu.

14.2.3 Źródła

Syllabus posiada referencje do uznanych standardów, które muszą zostać wykorzystane podczas przygotowywania materiałów szkoleniowych. Każdy standard musi zostać użyty w wersji cytowanej w

aktualnej wersji tego syllabusa. Inne publikacje, wzory czy standardy nie wspomniane w niniejszym syllabusie mogą również być używane, ale nie pojawią się na egzaminie.

14.3 Ćwiczenia praktyczne

Krótkie ćwiczenia praktyczne również powinny zostać włączone do szkolenia dla każdego z rozdziałów, gdzie wymaga się od kandydatów korzystania ze swojej wiedzy (K3 lub wyżej). Ćwiczenia i materiały powinny być oparte na celach nauczania oraz treści tematów znajdujących się w tym syllabusie.

15. Dodatek D – Zalecenia

Ze względu na fakt, że niniejsze zalecenia odnoszą się do różnych rozdziałów syllabusa, zostały one zebrane i spisane w jednym Dodatku. Poniższe punkty podlegają ocenie[11].

Lista dostarcza zestawu przydatnych zaleceń, które umożliwiają sprostanie wyzwaniom testowania i są zbudowane w oparciu o siedem podstawowych zasad testowania zaprezentowanych na poziomie podstawowym. Z założenia lista nie jest kompletna, zamiast tego zaprezentowano tutaj przykładowy zestaw lekcji wyniesionych, które powinny zostać wzięte pod uwagę. Dostawcy szkoleń powinni wybrać z niej zagadnienia, które są według nich najbardziej odpowiednie w kontekście nauczanego modułu.

15.1 Zalecenia dla przemysłu

Podczas wdrażania zaleceń w przemyśle stoi przed nami wiele wyzwań. Testerzy poziomu zaawansowanego powinni być w stanie przełożyć różne zalecenia opisane w tym syllabusie na ich organizacje, zespoły, zadania i moduły oprogramowania. Poniższa lista zawiera zestawienie zagadnień, dla których udowodniono, że mają negatywny wpływ na wydajność testowania. Z założenia lista nie wyczerpuje tego tematu.

- Tworzenie planów testów obejmujących wyłącznie testowanie funkcjonalne.

  Defekty nie dotyczą wyłącznie aspektów funkcjonalnych, czy działania pojedynczego użytkownika. Współdziałanie wielu użytkowników może mieć wpływ na testowany system.

- Zbyt mało testów konfiguracji.

  W sytuacji, gdy wiele rodzajów procesorów, systemów operacyjnych, maszyn wirtualnych, przeglądarek i urządzeń peryferyjnych tworzy wiele możliwych konfiguracji, ograniczenie testowania do tylko kilku z tych konfiguracji może doprowadzić do tego, że wiele potencjalnych defektów nie zostanie wykrytych.

- Odkładanie testowania przeciążającego i obciążeniowego na ostatnią chwilę.

  W wyniku przeprowadzenia testów przeciążających i obciążeniowych może się okazać, że wymagane są znaczące zmiany w testowanym oprogramowaniu (np. zmiany dotyczące architektury). Ze względu na fakt, że tego rodzaju zmiany mogą pociągać za sobą duże nakłady pracy, taka sytuacja może mieć bardzo negatywny wpływ na projekt, o ile tego rodzaju testy przeprowadzane są dopiero bezpośrednio przed wdrożeniem produkcyjnym.

- Brak testów dokumentacji.

[11] W oryginale jest „The items listed below are examinable”. [Przypis SJSI]

Użytkownikom dostarczane jest oprogramowanie wraz z towarzyszącą mu dokumentacją. W sytuacji, gdy dokumentacja nie pasuje do oprogramowania, użytkownik nie będzie mógł wykorzystać jego pełnego potencjału lub wręcz odrzuci całe oprogramowanie.

- Brak testów procedur instalacyjnych.

  Zarówno procedury instalacyjne, jak i procedury backupu i odtwarzania są testowane bardzo rzadko. Tymczasem są one bardziej krytyczne od samego oprogramowania; o ile oprogramowanie nie może być zainstalowane, nie będzie w ogóle używane.

- Naciskanie na ukończenie jednego zadania testowego zanim rozpoczęte zostanie kolejne.

  Pomimo iż w niektórych modelach cyklu życia oprogramowania położono nacisk na sekwencyjne wykonywanie zadań, w praktyce wiele z nich często musi być wykonywane jednocześnie (przynajmniej częściowo).

- Nieprawidłowa identyfikacja obszarów ryzyka.

  Niektóre z obszarów mogą być oznaczone jako obszary wysokiego ryzyka i przez to podlegać dokładniejszemu testowaniu. Tymczasem może się okazać, że obszary, w których zminimalizowano liczbę testów lub ich w ogóle nie wykonano, okażą się obszarami większego ryzyka niż estymowano na początku.

- Zbyt duża dokładność w odniesieniu do danych testowych i procedur testowych.

  Specyfikując zbyt dokładnie dane testowe oraz procedury testowe, tester może nie być zachęcony do sprawdzenia obszarów, które mogą zawierać wiele ukrytych defektów.

- Niezwracanie uwagi na wydające się z początku nieistotne osobliwości

  Obserwacje, czy rezultaty, które z początku mogą wydawać się nieistotne, często wskazują na ukryte defekty (jak góry lodowe, które tylko częściowo wystają ponad powierzchnię wody).

- Sprawdzanie, czy produkt robi to co powinien, a pomijanie sprawdzenia, czy nie robi tego, czego nie powinien.

  Ograniczając sprawdzenie do tego, co produkt powinien wykonywać, bardzo łatwo jest pominąć to, czego oprogramowanie nie powinno wykonywać (np. dodatkowe, niezaplanowane funkcje).

- Zestawy testowe rozumiane wyłącznie przez ich autorów.

  Testerzy mogą zmieniać obszary odpowiedzialności. Testerzy muszą móc przeczytać i zrozumieć testy wyspecyfikowane wcześniej. Tworzenie nieczytelnych i niezrozumiałych specyfikacji testów może mieć negatywny wpływ, ponieważ cele testowania mogą nie zostać zrozumiane lub test może być całkowicie usunięty.

- Wykonywanie testów wyłącznie przez interfejs dostępny dla użytkownika.

  Interfejs dostępny dla użytkownika końcowego nie jest jedynym interfejsem. Podczas przeprowadzania testów należy wziąć również pod uwagę komunikację pomiędzy wewnętrznymi procesami, wywołania *batch'owe* i inne zakłócenia, w których również mogą znajdować się defekty.

- Słabe raportowanie błędów oraz zarządzanie konfiguracją.

  Zarówno raportowanie i zarządzanie incydentami, jak i zarządzanie konfiguracją są niezwykle ważne na drodze do osiągnięcia sukcesu całego projektu (który zawiera rozwój i testowanie). Skuteczny tester to tester, który doprowadza do usunięcia wielu defektów, a nie tester, który znajduje wiele defektów, jednak jego sposób raportowania nie jest wystarczający, co w konsekwencji powoduje sytuację, że znalezione defekty nie są usuwane.

- Dodawanie wyłącznie testów regresywnych.

  Rozwój zestawów testowych nie powinien być ograniczony wyłącznie do sprawdzania, czy nie pojawiły się defekty w ramach testów regresywnych. Kod jest rozwijany i może się okazać, że wymagane jest utworzenie dodatkowych testów do pokrycia nowych funkcjonalności i wykonania testów regresywnych w innych obszarach oprogramowania.

- Nietworzenie notatek do wykorzystania podczas kolejnych czynności testowych.

  Zadania testownie nie kończą się, gdy oprogramowanie zostanie dostarczone do użytkownika lub przekazane na rynek. Istnieje duże prawdopodobieństwo, że powstanie nowa wersja, stąd wiedza powinna zostać zgromadzona i przekazana testerom odpowiedzialnym za kolejne czynności testowe.

- Próba zautomatyzowania wszystkich testów.

  Automatyzacja może wydawać się dobrym pomysłem, jednak jest ona projektem rozwojowym samym w sobie. Nie wszystkie testy powinny być zautomatyzowane: niektóre z nich mogą być szybciej wykonywane w sposób manualny niż automatyczny.

- Oczekiwanie, że wszystkie testy manualne zostaną wykonane ponownie.

  Podczas ponownego wykonywania testów manualnych, nie można oczekiwać, że wszystkie testy zostaną powtórzone. Uwaga testerów zostanie zachwiana i zaczną się oni skupiać na testowaniu niektórych obszarów, świadomie bądź nieświadomie.

- Używanie narzędzi rejestrująco-odtwarzających GUI, aby zredukować koszty tworzenia testów.

  Wykorzystanie narzędzi rejestrująco-odtwarzających GUI łączy się z dużą inwestycją początkową i powinno być wynikiem zdefiniowanej strategii, po zidentyfikowaniu i ocenie wszystkich powiązanych z tym kosztów.

- Oczekiwanie, że w testach regresywnych zostanie znalezione wiele nowych błędów.

  Z założenia, podczas testów regresywnych nie zostaje znaleziony duży odsetek błędów, a to dlatego, że te testy są wykonywane po raz kolejny (np. były już wykonanywane dla wcześniejszych wersji oprogramowania) a defekty zostały znalezione już w poprzednich uruchomieniach testów. Nie oznacza to jednak, że należy z nich w pełni zrezygnować. Należy jednak pamiętać, że ich efektywność (zdolność do znajdowania nowych defektów) jest niższa niż efektywność innych testów.

- Wysoki poziom pokrycia, w założeniu, że tylko proste liczby mogą inspirować.

  Pokrycie kodu i miary (liczby i grafy) mogą wyglądać interesująco z punktu widzenia kierownictwa, jednakże mogą one nie odzwierciedlać efektywności i słuszności przeprowadzonych testów. Przykład: 100% jest „ładnym" celem dla pokrycia kodu, ale czy jest realistyczne, czy jest adekwatne (np. czy chodzi o pokrycie instrukcji, warunków, pokrycie kombinacji warunków w decyzjach)?

- Usuwanie testów z zestawów testów regresywnych tylko dlatego, że ich wykonanie nie zwiększa pokrycia testowego.

  Część testów regresywnych może (powinna) być usuwana, a część dodawana. Powodem usunięcia testów z zestawu testów regresywnych nie powinien być fakt, czy dany test wpływa na zwiększenie pokrycia testów (może się zdarzyć, że dany test nie wpływa na pokrycie, jednak zwiększa skuteczność testowania np. poprzez sprawdzenie oprogramowania pod kątem specyficznego rodzaju defektu). Przykład: pokrycie kodu nie jest jedynym rodzajem pokrycia; testy mogą być tworzone z innych powodów (np. wprowadzanie specyficznych wartości lub sekwencji wydarzeń) niż zwykłe pokrycie.

- Osiągnięcie pewnego poziomu pokrycia określa wydajność testerów.

  Pokrycie jest miarą kompletności, a nie wydajności lub efektywności personelu. Do oceny efektywności testerów w organizacji i projekcie można użyć innych miar. Użycie tych miar musi być jednak starannie przemyślane, aby uniknąć niezamierzonych efektów („dysfunkcje").

- Całkowite pomijanie pokrycia.

  Dostępne są różne rodzaje pokrycia (np. instrukcji kodu, warunków, modułów, funkcji itp), a wysiłek, który należy włożyć, aby uzyskać te miary może być znaczący. Nie jest to jednak powodem, aby całkowicie rezygnować ze zbierania informacji o pokryciu, jako że mogą one być bardzo przydatne podczas wykonywania zadań testowych.

Wiele z powyższych punktów odnosi się do konkretnych sekcji tego syllabusa.

Podczas wprowadzania metryk i praktyk testowych w organizacji, nie powinno się kierować wyłącznie zagadnieniami wymienionymi powyżej, lecz również rozważyć dodatkowe źródła informacji, takie jak:

- Syllabus poziomu podstawowego i zaawansowanego
- Książki wylistowane na liście odnośników w tym syllabusie
- Modele referencyjne tj. te, o których była mowa w rozdziale 8.3.