

Certyfikowany tester ISTQB®

**Sylabus specjalistyczny
Inżynier Automatyzacji Testów**

wersja 1.0. (2016)
(1.0.0.1 PL)

International Software Testing Qualifications Board®
© Stowarzyszenie Jakości Systemów Informatycznych



Prawa autorskie

Kopiowanie całości lub fragmentów niniejszego dokumentu jest dozwolone pod warunkiem wskazania źródła.

Copyright © International Software Testing Qualifications Board (zwana dalej ISTQB®).

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

Tłumaczenie z języka angielskiego wersji beta – BTInfo Biuro Tłumaczeń Informatycznych Przyłuccy Sp. j.

Wszelkie prawa zastrzeżone

Autorzy niniejszym przenoszą autorskie prawa majątkowe na ISTQB®. Autorzy (jako obecni posiadacze autorskich praw majątkowych) oraz ISTQB® (jako przyszły posiadacz autorskich praw majątkowych) uzgodnili następujące warunki korzystania z dokumentu:

Każdy akredytowany dostawca szkoleń może wykorzystywać ten sylabus jako podstawę dla szkolenia, o ile zachowane są informacje o autorach i ISTQB® jako źródle i właścicielach praw autorskich do sylabusa. Powoływanie się na niniejszy sylabus we wszelkich materiałach reklamowych i promocyjnych dozwolone jest dopiero po uzyskaniu oficjalnej akredytacji materiałów szkoleniowych udzielonej przez uznaną przez ISTQB® Radę Krajową (w przypadku Polski przez Stowarzyszenie Jakości Systemów Informatycznych).

Każda osoba lub grupa osób może używać tego sylabusa jako podstawy dla artykułów i książek, jeśli autorzy i ISTQB® są wskazani jako źródło i właściciele praw autorskich do sylabusa. Każde inne użycie sylabusa jest zabronione bez wcześniejszego uzyskania zgody ISTQB®.

Każda Rada Krajowa uznawana przez ISTQB® może przetłumaczyć ten sylabus pod warunkiem, że powieli i opublikuje wyżej wymienioną informację o prawach autorskich w przetłumaczonej wersji sylabusa.

Grupa Robocza ds. Automatyzacji Testów na Poziomie Zaawansowanym (Advanced Level Test Automation Working Group): Bryan Bakker, Graham Bath, Armin Born, Mark Fewster, Jani Haukinen, Judy McKay, Andrew Pollner, Raluca Popescu, Ina Schieferdecker (2016).

Historia zmian

Wersja	Data	Uwagi
1.0.0.1 PL	27.02.2024 r.	Przegląd i korekta tłumaczenia. Wprowadzenie nowego oznaczenia numeru wersji sylabusa. Zmiana tłumaczenia słów kluczowych w rozdziale 3: z „tworzenie skryptów liniowych” na „technika skryptów liniowych”, z „tworzenie skryptów sterowanych przez proces” na „technika skryptów sterowanych przez proces”, z „tworzenie skryptów ustrukturyzowanych” na „technika skryptów ustrukturalizowanych”. Zmiana tłumaczenia z „przypadki testowe dla testów automatycznych” na „zautomatyzowane przypadki testowe”, „zestaw testów” na „zestaw testowy”.
1.0.	13 czerwca 2022 r.	Rys. 4. Synchronizacja procesów wytwarzania TAS i SUT, dodano strzałkę (przykład nr 2)
Sylabus 2016	20 stycznia 2022 r.	Aktualizacja terminów, przegląd
Sylabus 2016	24 lutego 2021 r.	Przegląd edytorski
Sylabus 2016	21 października 2016 r.	Wersja zatwierdzona do publikacji przez Zgromadzenie Ogólne ISTQB®

Spis treści

Prawa autorskie	2
Historia zmian	3
0. Wstęp	7
0.1. Cel	7
0.2. Zakres dokumentu	7
0.2.1. Zagadnienia uwzględnione w dokumencie	7
0.2.2. Zagadnienia nieuwzględnione w dokumencie	7
0.3. Certyfikowany tester, poziom specjalistyczny — inżynier automatyzacji testów	8
0.3.1. Oczekiwania	8
0.3.2. Wymagania związane z przystąpieniem do egzaminu i odnawianiem uprawnień	8
0.3.3. Poziom wiedzy	8
0.3.4. Egzamin	8
0.3.5. Akredytacja	9
0.4. Części normatywne i informacyjne	9
0.5. Poziom szczegółowości	9
0.6. Struktura sylabusu	9
0.7. Terminy, definicje i skróty	9
1. Wprowadzenie do automatyzacji testów i jej cele — 30 min	11
1.1. Cel automatyzacji testów	12
1.2. Czynniki sukcesu w zakresie automatyzacji testów	13
2. Przygotowanie do automatyzacji testów — 165 min	16
2.1. Czynniki związane z systemem podlegającym testowaniu wpływające na automatyzację testów	17
2.2. Ocena i wybór narzędzi	18
2.3. Projektowanie pod kątem testowalności i automatyzacji	20
3. Ogólna architektura testów automatycznych — 270 min	22
3.1. Wprowadzenie do ogólnej architektury testów automatycznych (gTAA)	23
3.1.1. Omówienie ogólnej architektury testów automatycznych (gTAA)	24
3.1.2. Warstwa generowania testów	25
3.1.3. Warstwa definiowania testów	26
3.1.4. Warstwa wykonywania testów	26
3.1.5. Warstwa adaptacji testów	26
3.1.6. Zarządzanie konfiguracją TAS	27
3.1.7. Zarządzanie projektem w zakresie TAS	27
3.1.8. Obsługa zarządzania testami na poziomie TAS	27
3.2. Projektowanie architektury automatyzacji testów (TAA)	27
3.2.1. Wprowadzenie do projektowania TAA	27
3.2.2. Podejścia do automatyzacji przypadków testowych	31
3.2.3. Uwarunkowania techniczne związane z systemem podlegającym testowaniu (SUT)	36
3.2.4. Uwarunkowania związane z procesami wytwarzania oprogramowania i zapewnienia jakości	37
3.3. Tworzenie rozwiązania dla testów automatycznych (TAS)	38
3.3.1. Wprowadzenie do procesu tworzenia TAS	38

3.3.2.	Kompatybilność TAS z systemem podlegającym testowaniu (SUT).....	39
3.3.3.	Synchronizacja TAS z systemem podlegającym testowaniu (SUT).....	39
3.3.4.	Projektowanie TAS pod kątem ponownego wykorzystania	42
3.3.5.	Obsługa różnych systemów docelowych	43
4.	Ryzyka i sytuacje awaryjne związane z wdrożeniem — 150 min	44
4.1.	Wybór podejścia do automatyzacji testów oraz planowanie wdrożenia/wydania na rynek	45
4.1.1.	Projekt pilotażowy	45
4.1.2.	Wdrożenie	46
4.1.3.	Wdrożenie TAS w kontekście cyklu życia oprogramowania	47
4.2.	Strategie oceny i łagodzenia ryzyka	47
4.3.	Pielęgnacja rozwiązań dla automatyzacji testów	49
4.3.1.	Rodzaje pielęgnacji	49
4.3.2.	Zakres i podejście	49
5.	Raporty i metryki dotyczące automatyzacji testów — 165 min	52
5.1.	Wybór metryk dotyczących rozwiązania dla testów automatycznych (TAS)	53
5.2.	Implementacja pomiarów	56
5.3.	Logowanie danych dotyczących rozwiązania dla testów automatycznych (TAS) i systemu podlegającego testowaniu (SUT).....	57
5.4.	Raportowanie z testów automatycznych.....	58
6.	Przeniesienie testowania manualnego do środowiska zautomatyzowanego — 120 min	60
6.1.	Kryteria automatyzacji.....	61
6.2.	Zidentyfikowanie kroków niezbędnych do implementacji zautomatyzowanych testów regresji	65
6.3.	Czynniki, które należy wziąć pod uwagę przy automatyzowaniu testów nowych funkcjonalności (cech)	67
6.4.	Czynniki, które należy wziąć pod uwagę przy implementacji automatycznych testów potwierdzających	68
7.	Weryfikowanie rozwiązania dla testów automatycznych (TAS) — 120 min.....	69
7.1.	Weryfikowanie komponentów zautomatyzowanego środowiska testowego	70
7.2.	Weryfikowanie zestawu testów automatycznych	72
8.	Ciągłe doskonalenie — 150 min.....	74
8.1.	Możliwości wprowadzania udoskonaleń w zakresie automatyzacji testów	75
8.2.	Planowanie implementacji udoskonaleń w zakresie automatyzacji testów	77
9.	Bibliografia	79
9.1.	Standardy	79
9.2.	Dokumenty ISTQB®	80
9.3.	Znaki towarowe	80
9.4.	Książki	80
9.5.	Materiały źródłowe w Internecie.....	81
10.	Uwagi dla dostawców szkoleń	82
10.1.	Czas trwania szkolenia	82
10.2.	Praktyczne ćwiczenia do wykonania w miejscu pracy.....	82
10.3.	Zasady dotyczące e-nauczania	82
11.	Indeks	83

Podziękowania

Niniejszy dokument został opracowany przez podstawowy zespół Grupy Roboczej Poziomu Zaawansowanego (Advanced Level Working Group) działającej w ramach ISTQB®.

Członkowie zespołu pragną podziękować zespołowi przeglądających oraz wszystkim Radom Krajowym za sugestie i wskazówki.

W chwili zakończenia prac nad sylabusem poziomu zaawansowanego do tego modułu w skład Grupy Roboczej ds. Automatykacji Testów na Poziomie Zaawansowanym wchodziły następujące osoby: Bryan Bakker, Graham Bath (Przewodniczący Grupy Roboczej Poziomu Zaawansowanego), Armin Beer, Inga Birthe, Armin Born, Alessandro Collino, Massimo Di Carlo, Mark Fewster, Mieke Gevers, Jani Haukinen, Skule Johansen, Eli Margolin, Judy McKay (Wiceprzewodnicząca Grupy Roboczej Poziomu Zaawansowanego), Kateryna Nesmyelova, Mahantesh (Monty) Pattan, Andrew Pollner (Przewodniczący Grupy Roboczej Automatykacji Testów na Poziomie Zaawansowanym), Raluca Popescu, Ioana Prundaru, Riccardo Rosci, Ina Schieferdecker, Gil Shekel oraz Chris Van Bael.

Autorami sylabusu są następujący członkowie podstawowego zespołu: Andrew Pollner (Przewodniczący), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu oraz Ina Schieferdecker.

W procesie przeglądu uczestniczyły, zgłaszały uwagi i głosowały nad niniejszym sylabusem następujące osoby (w kolejności alfabetycznej): Armin Beer, Tibor Csondes, Massimo Di Carlo, Chen Geng, Cheryl George, Kari Kakkonen, Jen Leger, Singh Manku, Ana Paiva, Raluca Popescu, Meile Posthuma, Darshan Preet, Ioana Prundaru, Stephanie Ulrich, Erik van Veenendaal oraz Rahul Verma.

Niniejszy dokument został formalnie zatwierdzony do publikacji przez Zgromadzenie Ogólne ISTQB® w dniu 21 października 2016 r.

Koordinację prac tłumaczenia na język polski wykonały następujące osoby: Damian Brzęczek, Jan Sabak. W procesie przeglądu tłumaczenia uczestniczyli: Bartosz Walter, Damian Brzęczek, Michał Krześlak.

Przegląd tłumaczenia (2021 r.): Monika Petri-Starego (kierownik zespołu), Adam Ścierański, Adam Roman, Damian Brzęczek, Grzegorz Holak. Wykonanie grafik: Monika Krzyszczanowicz.

Redakcja tekstu i korekta tłumaczenia (2024): Monika Petri-Starego, przegląd Adam Roman i Lucjan Stapp.

0. Wstęp

0.1. Cel

Niniejszy sylabus stanowi podstawę egzaminu International Software Testing Qualifications Board w zakresie inżynierii automatyzacji testów na poziomie specjalistycznym. ISTQB® udostępnia ten sylabus:

- Radom Krajowym w celu przetłumaczenia na języki lokalne i dokonania akredytacji dostawców szkoleń (Rady Krajowe mogą dostosować sylabus do potrzeb danego języka oraz dodawać odwołania do literatury w celu uwzględnienia publikacji lokalnych),
- organom certyfikacyjnym w celu sformułowania pytań egzaminacyjnych w językach lokalnych (dostosowanych do celów nauczania poszczególnych modułów),
- dostawcom szkoleń w celu opracowania materiałów dydaktycznych i określenia odpowiednich metod nauczania,
- kandydatom ubiegającym się o certyfikat w celu przygotowania się do egzaminu (w ramach szkoleń lub samodzielnie),
- międzynarodowej społeczności specjalistów w dziedzinie inżynierii oprogramowania i systemów informatycznych w celu rozwijania zawodu testera oprogramowania i systemów oraz opracowywania książek i artykułów.

ISTQB® może również zezwolić innym podmiotom na korzystanie z niniejszego sylabusa do innych celów pod warunkiem uzyskania uprzedniej pisemnej zgody od ISTQB®. Dokument ten może być kopiowany w całości lub w części, jeśli zostanie podane źródło.

0.2. Zakres dokumentu

0.2.1. Zagadnienia uwzględnione w dokumencie

W niniejszym dokumencie opisano zadania inżyniera automatyzacji testów (ang. *Test Automation Engineer* — TAE) związane z projektowaniem, tworzeniem i utrzymaniem rozwiązań do automatyzacji testów. Autorzy skupili się na pojęciach, metodach, narzędziach i procesach związanych z automatyzacją dynamicznych testów funkcjonalnych oraz na powiązaniach między tymi testami a procesami zarządzania testami, zarządzania konfiguracją, zarządzania defektami, wytwarzania oprogramowania i zapewnienia jakości oprogramowania.

Opisane metody mają ogólne zastosowanie w różnych modelach cyklu życia oprogramowania (np. model zwinny, sekwencyjny, przyrostowy lub iteracyjny), dla różnych typów systemów oprogramowania (np. systemów wbudowanych, rozproszonych lub mobilnych) oraz różnych typów testów (tj. testów funkcjonalnych i testów нефunkcjonalnych).

0.2.2. Zagadnienia nieuwzględnione w dokumencie

W niniejszym sylabusie dotyczącym inżynierii automatyzacji testów nie zostały uwzględnione następujące zagadnienia:

- zarządzanie testami, automatyczne tworzenie specyfikacji testów i automatyczne generowanie testów,
- zadania kierownika automatyzacji testów (TAM) związane z planowaniem, nadzorowaniem i dostosowywaniem procesów tworzenia i rozwoju rozwiązań do automatyzacji testów,
- specyfikacja automatyzacji testów нефunkcjonalnych (np. kwestie wydajności),
- automatyzacja analizy statycznej (np. analizy podatności) i narzędzi do testów statycznych,
- nauczanie metod inżynierii oprogramowania i zasad programowania (np. w zakresie doboru stosowanych standardów czy określania kwalifikacji niezbędnych do realizacji rozwiązań do automatyzacji testów),

- nauczanie technologii oprogramowania (np. w zakresie wyboru technik tworzenia skryptów w celu implementacji rozwiązań do automatyzacji testów),
- wybór produktów i usług związanych z testowaniem oprogramowania (np. dobór produktów i usług używanych w ramach rozwiązania dla testów automatycznych).

0.3. Certyfikowany tester, poziom specjalistyczny — inżynier automatyzacji testów

0.3.1. Oczekiwania

Certyfikacja na poziomie specjalistycznym jest przeznaczona dla osób, które chcą poszerzyć wiedzę i umiejętności zdobyte na poziomie podstawowym oraz dalej rozwijać specjalistyczną wiedzę w jednym lub kilku konkretnych obszarach. Moduły oferowane na poziomie specjalistycznym obejmują szeroki zakres tematów związanych z testowaniem.

Inżynier automatyzacji testów musi wykazywać się zarówno obszerną wiedzą ogólną w zakresie testowania, jak i dogłębną znajomością szczególnej dziedziny, jaką jest automatyzacja testów. Określenie „dogłębna znajomość” oznacza posiadanie wiedzy na temat teorii i praktyki automatyzacji testów, która jest wystarczająca do wywierania wpływu na kierunek działań organizacji i/lub sposób realizacji projektu na etapach projektowania, tworzenia i utrzymania rozwiązań do automatyzacji testów w obszarze testów funkcjonalnych.

Cele biznesowe tego modułu opisano w dokumencie zatytułowanym „Przegląd modułów poziomu zaawansowanego” [ISTQB-AL-Modules].¹

0.3.2. Wymagania związane z przystąpieniem do egzaminu i odnawianiem uprawnień

Aby przystąpić do egzaminu certyfikacyjnego na inżyniera automatyzacji testów na poziomie specjalistycznym, kandydat musi posiadać certyfikat ISTQB® poziomu podstawowego [ISTQB-CTFL].

0.3.3. Poziom wiedzy

Cele nauczania realizowane przez niniejszy sylabus wymieniono na początku każdego rozdziału, aby umożliwić ich jednoznaczne zidentyfikowanie. Znajomość poszczególnych zagadnień przewidzianych w sylabusie będzie sprawdzana na egzaminie zgodnie z przypisanymi celami nauczania.

Poziomy poznawcze (tzw. poziomy „K”) przyporządkowane do poszczególnych celów nauczania opisano na stronie internetowej ISTQB® [ISTQB-Web].

0.3.4. Egzamin

Egzamin umożliwiający uzyskanie certyfikatu na poziomie Specjalista jest oparty na niniejszym sylabusie oraz sylabusie poziomu podstawowego [ISTQB-FL]. Przy udzielaniu odpowiedzi na pytania egzaminacyjne może być konieczne skorzystanie z materiału obejmującego więcej niż jeden rozdział powyższych sylabusów. Na stronie internetowej ISTQB® [ISTQB-Web] są dostępne informacje przydatne przy zdawaniu egzaminów.

¹ W oryginale ten sylabus jest określony jako Advanced Level, a po wydaniu tego sylabusa certyfikacja TAE przeszła do ścieżki Specialist.

0.3.5. Akredytacja

Rada Krajowa ISTQB® może dokonywać akredytacji dostawców szkoleń, którzy oferują materiały dydaktyczne zgodne z niniejszym sylabusem.

0.4. Części normatywne i informacyjne

Przedmiotem egzaminu może być treść części normatywnych sylabusa. Są to:

- cele nauczania,
- słowa kluczowe.

Pozostałe części sylabusa mają charakter informacyjny i służą dokładniejszemu omówieniu celów nauczania.

0.5. Poziom szczegółowości

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

- cele nauczania dla poszczególnych obszarów wiedzy, opisujące efekty kształcenia o charakterze poznawczym oraz sposób myślenia, który należy osiągnąć (część normatywna),
- wykaz informacji przekazywanych w procesie kształcenia, w tym opis najważniejszych pojęć, listę źródeł (takich jak uznane publikacje lub standardy) oraz odniesienia do dodatkowych źródeł, jeśli są wymagane (część informacyjna).

Zawartość sylabusa nie stanowi opisu całego obszaru wiedzy związanego z inżynierią automatyzacji testów. Odzwierciedla ona jedynie poziom szczegółowości, jaki należy uwzględnić w akredytowanym szkoleniu na poziomie specjalistycznym.

0.6. Struktura sylabusa

Dokument podzielono na osiem głównych rozdziałów. Nagłówek najwyższego poziomu zawiera informację o czasie trwania szkolenia dotyczącego treści danego rozdziału, na przykład:

3. Ogólna architektura testów automatycznych	270 min
--	---------

Powyższa informacja oznacza, że na przekazanie wiedzy zawartej w rozdziale 3. przeznaczono 270 minut. Na początku każdego rozdziału wymieniono szczegółowe cele nauczania.

0.7. Terminy, definicje i skróty

Wiele terminów stosowanych w literaturze z dziedziny oprogramowania jest używanych zamiennie. Definicje terminów stosowanych w niniejszym sylabusie poziomu Specjalista są dostępne w opublikowanym przez ISTQB® Słowniku terminów testowych ISTQB® [ISTQB-Glossary].

Ponadto w słowniku zdefiniowano poszczególne słowa kluczowe umieszczone na początku każdego rozdziału niniejszego sylabusa poziomu Specjalista.

W dokumencie używane są następujące akronimy:

CLI	Command Line Interface — interfejs wiersza poleceń
EMTE	Equivalent Manual Test Effort — ekwiwalent pracochłonności testowania manualnego
gTAA	Generic Test Automation Architecture — ogólna architektura testów automatycznych (będąca wzorem dla rozwiązań do automatyzacji testów)
GUI	Graphical User Interface — graficzny interfejs użytkownika
SUT	System Under Test — system podlegający testowaniu (patrz także: przedmiot testów)
TAA	Test Automation Architecture — architektura testów automatycznych (praktyczna realizacja gTAA określająca architekturę rozwiązania dla automatyzacji testów).

TAE	Test Automation Engineer — inżynier automatyzacji testów (osoba odpowiedzialna za zaprojektowanie TAA oraz implementację, utrzymanie i rozwój techniczny powstałego w rezultacie TAS)
TAF	Test Automation Framework — struktura do testów automatycznych (środowisko niezbędne do automatyzacji testów, obejmujące między innymi narzędzia testowe oraz artefakty takie jak biblioteki testowe)
TAM	Test Automation Manager — kierownik automatyzacji testów (osoba odpowiedzialna za planowanie i nadzór nad procesem tworzenia i rozwoju rozwiązania do automatyzacji testów)
TAS	Test Automation Solution — rozwiązanie dla testów automatycznych (realizacja/implementacja TAA obejmująca między innymi narzędzia testowe oraz artefakty takie jak biblioteki testowe)
UI	User Interface — interfejs użytkownika
SDLC	Software Development Lifecycle — cykl życia oprogramowania.

1. Wprowadzenie do automatyzacji testów i jej cele — 30 min.

Słowa kluczowe

testowanie przy użyciu interfejsów: API, CLI, GUI, system podlegający testowaniu, architektura testów automatycznych, struktura do testów automatycznych, strategia automatyzacji testów, automatyzacja testów, skrypt testowy, testalia

Cele nauczania — wprowadzenie do automatyzacji testów i jej cele

1.1 Cel automatyzacji testów

ALTA-E-1.1.1 (K2) Kandydat potrafi objaśnić cele, zalety, wady i ograniczenia automatyzacji testów.

1.2 Czynniki sukcesu w zakresie automatyzacji testów

ALTA-E-1.2.1 (K2) Kandydat potrafi wskazać czynniki decydujące o sukcesie projektu automatyzacji testów.

1.1 Cel automatyzacji testów

Automatyzacja testów (obejmująca także wykonywanie testów automatycznych) w kontekście testowania oprogramowania oznacza proces obejmujący jedno lub kilka następujących działań:

- kontrolowanie i konfigurowanie warunków początkowych testów przy użyciu specjalnie w tym celu opracowanych narzędzi,
- wykonywanie testów,
- porównywanie wyników rzeczywistych z oczekiwanymi.

W celu zminimalizowania wzajemnego wpływu na siebie, zaleca się oddzielenie oprogramowania służącego do wykonywania testów od systemu podlegającego testowaniu (SUT). Dopuszcza się jednak wyjątki od tej zasady — przykładem mogą być systemy wbudowane, w przypadku których konieczna jest integracja oprogramowania testowego z systemem podlegającym testowaniu.

Automatyzacja testów ma z założenia ułatwić konsekwentne, wielokrotne wykonywanie dużej liczby przypadków testowych w różnych wersjach systemu podlegającego testowaniu i/lub na różnych środowiskach. Nie oznacza to jednak, że jest ona jedynie mechanizmem do wykonywania zestawu testów bez interakcji z człowiekiem. Ważnym jej elementem jest również proces projektowania testaliów, do których zaliczają się:

- oprogramowanie
- dokumentacja,
- przypadki testowe,
- środowiska testowe,
- dane testowe.

Testalia są niezbędne do wykonywania czynności testowych obejmujących:

- implementowanie zautomatyzowanych przypadków testowych,
- monitorowanie i kontrolowanie wykonywania testów automatycznych,
- interpretowanie, raportowanie oraz rejestrowanie rezultatów testów automatycznych.

W ramach automatyzacji testów mogą być stosowane różne podejścia do interakcji z systemem podlegającym testowaniu:

- testowanie za pośrednictwem publicznych interfejsów do klas, modułów lub bibliotek systemu podlegającego testowaniu (testowanie przy użyciu API),
- testowanie za pośrednictwem interfejsu użytkownika systemu podlegającego testowaniu (np. testowanie przy użyciu GUI lub testowanie przy użyciu CLI),
- testowanie za pośrednictwem usługi lub protokołu.

Wśród celów automatyzacji testów można wymienić:

- zwiększenie efektywności wykonywania testów,
- zapewnienie szerszego pokrycia funkcji,
- obniżenie całkowitych kosztów testowania,
- wykonywanie testów, których testerzy nie mogą wykonać manualnie,
- skrócenie okresu wykonywania testów,
- zwiększenie częstotliwości testów lub skrócenie czasu trwania cykli testowych.

Do zalet automatyzacji testów można zaliczyć:

- możliwość wykonywania większej liczby testów w przypadku każdej wersji,
- możliwość tworzenia testów, których nie da się wykonać manualnie (np. testów wykonywanych w czasie rzeczywistym, testów zdalnych czy testów równoległych),
- możliwość wykonywania bardziej złożonych testów,
- większą szybkość wykonywania testów,
- ograniczenie podatności testów na błędy operatora,
- skuteczniejsze i efektywniejsze wykorzystanie zasobów testowych,
- szybsze uzyskiwanie informacji zwrotnych na temat jakości oprogramowania,
- wyższą niezawodność systemu (np. powtarzalność, spójność),
- większą spójność testów.

Wady automatyzacji testów to między innymi:

- dodatkowe koszty,
- początkowe nakłady inwestycyjne związane ze skonfigurowaniem rozwiązania dla testów automatycznych (TAS),
- konieczność stosowania dodatkowych technologii,
- konieczność posiadania przez zespół umiejętności w zakresie programowania i automatyzacji,
- wymóg ciągłego utrzymania TAS,
- ryzyko odwrócenia uwagi od celów testowania (np. w wyniku skoncentrowania się na automatyzacji przypadków testowych kosztem wykonywania testów),
- potencjalny wzrost złożoności testów,
- niebezpieczeństwo wprowadzenia dodatkowych błędów w wyniku automatyzacji.

Wśród ograniczeń związanych z automatyzacją testów należy wymienić:

- brak możliwości zautomatyzowania wszystkich testów manualnych,
- możliwość sprawdzania jedynie rezultatów dających się zinterpretować maszynowo,
- możliwość sprawdzania tylko tych rzeczywistych rezultatów, które mogą zostać zweryfikowane przez automatyczną wyrocznię testową,
- brak możliwości zastąpienia testów eksploracyjnych.

1.2 Czynniki sukcesu w automatyzacji testów

Przedstawione poniżej czynniki sukcesu dotyczą takich projektów automatyzacji testów, które są już realizowane, w związku z czym skupiono się na czynnikach wpływających na długofalowe powodzenie projektu. Nie uwzględniono natomiast czynników wpływających na powodzenie projektów automatyzacji testów znajdujących się w fazie pilotażowej.

Poniżej opisano najważniejsze czynniki sukcesu związane z automatyzacją testów:

Architektura testów automatycznych (TAA)

Architektura testów automatycznych (ang. *Test Automation Architecture* — TAA) jest bardzo ściśle związana z architekturą testowanego oprogramowania. W związku z tym należy jednoznacznie wskazać, które wymagania funkcjonalne i нефункционалне ma wspierać — zwykle są to wymagania uznawane za najważniejsze.

Architekturę testów automatycznych często projektuje się pod kątem utrzymywalności, wydajności i łatwości nauki (szczegółowe informacje na temat tych i innych cech нефункционалных zawiera norma ISO/IEC 25000:2014). W związku z tym wskazane jest zaangażowanie inżynierów oprogramowania, którzy znają architekturę systemu podlegającego testowaniu.

Testowalność systemu podlegającego testowaniu (SUT)

System podlegający testowaniu musi być zaprojektowany pod kątem testowalności, która wspiera zautomatyzowane testowanie. W przypadku testowania przy użyciu GUI może to oznaczać, że system podlegający testowaniu powinien w jak największym stopniu odseparować interakcję i dane GUI od wyglądu interfejsu graficznego. Z kolei w przypadku testowania przy użyciu API może to oznaczać konieczność udostępnienia większej liczby klas lub modułów (bądź CLI) w sposób umożliwiający ich przetestowanie.

W pierwszej kolejności należy zająć się testowalnymi elementami systemu podlegającego testowaniu. Ogólnie rzecz biorąc, kluczowym czynnikiem sukcesu związanym z automatyzacją testów jest zwykle łatwość implementacji zautomatyzowanych skryptów testowych. Aby osiągnąć ten cel i pomyślnie zaprezentować dowód słuszności koncepcji, inżynier automatyzacji testów (TAE) musi zidentyfikować moduły lub komponenty systemu podlegającego testowaniu dające się łatwo przetestować z wykorzystaniem automatyzacji i zacząć od nich.

Strategia automatyzacji testów

Niezbędna jest praktyczna i spójna strategia automatyzacji testów, uwzględniająca kwestie utrzymywaności i spójności systemu podlegającego testowaniu.

Nie zawsze można zastosować strategię automatyzacji testów w ten sam sposób w odniesieniu do starych i nowych elementów systemu podlegającego testowaniu. Dlatego przy tworzeniu strategii należy wziąć pod uwagę koszty, korzyści i ryzyko związane z jej stosowaniem do różnych obszarów kodu.

Należy również rozważyć przetestowanie GUI oraz API przy użyciu zautomatyzowanych przypadków testowych w celu sprawdzenia spójności rezultatów.

Struktura do testów automatycznych (TAF)

Łatwa w użyciu, dobrze udokumentowana i utrzymywano struktura do testów automatycznych (ang. *Test Automation Framework — TAF*) pozwala zapewnić spójne podejście do automatyzacji testów.

Aby stworzyć łatwą w użyciu i w utrzymaniu strukturę do testów automatycznych należy wykonać następujące czynności:

- Implementacja mechanizmów raportowania: raporty z testów powinny dostarczać informacji na temat jakości systemu podlegającego testowaniu (test zaliczony, niezaliczony, zakończony błędem, niewykonany lub anulowany, informacje statystyczne itd.). Informacje zawarte w raportach powinny umożliwiać zaangażowanym testerom, kierownikom testów, programistom, kierownikom projektu i innym interesariuszom uzyskanie ogólnego obrazu jakości systemu.
- Łatwe diagnozowanie problemów. Poza wykonywaniem testów i rejestrowaniem danych, TAF powinna również umożliwiać łatwe diagnozowanie przyczyn niezaliczonych testów. Test może nie zostać zaliczony na skutek:
 - awarii wykrytych w systemie podlegającym testowaniu,
 - awarii wykrytych w rozwiązaniu dla testów automatycznych (TAS),
 - problemów związanych z samymi testami lub środowiskiem testowym.
- Poświęcenie należytej uwagi środowisku testowemu. Funkcjonowanie narzędzi testowych zależy od spójności środowiska testowego. W przypadku testowania automatycznego konieczne jest stworzenie wydzielonego środowiska testowego, ponieważ brak kontroli nad tym środowiskiem i danymi testowymi może doprowadzić do sytuacji, w której konfiguracja testowa nie spełni wymagań związanych z wykonywaniem testów, a uzyskane rezultaty będą fałszywe.
- Dokumentowanie zautomatyzowanych przypadków testowych. Należy jednoznacznie określić cele automatyzacji testów: które elementy aplikacji mają być testowane i w jakim stopniu, które atrybuty (funkcjonalne i niefunkcjonalne) mają być przedmiotem testów itd. Następnie należy w przejrzysty sposób opisać i udokumentować przyjęte założenia.
- Śledzenie testów automatycznych. Struktura do testów automatycznych musi umożliwiać inżynierowi automatyzacji testów śledzenie powiązań między poszczególnymi krokami a przypadkami testowymi.
- Łatwość pielęgnacji. Zautomatyzowane przypadki testowe powinny być łatwe w utrzymaniu, tak, aby nie pochłaniało ono istotnej części nakładu pracy związanego z automatyzacją testów. Ponadto, nakład pracy na pielęgnację musi być proporcjonalny do zakresu zmian wprowadzanych w systemie podlegającym testowaniu, co z kolei oznacza, że przypadki testowe muszą być łatwe do analizowania, modyfikowania i rozszerzania. Inną ważną kwestią jest zapewnienie wysokiego stopnia ponownego wykorzystania testaliów w testach automatycznych, co pozwoli ograniczyć liczbę elementów wymagających zmian.

- Potrzeba ciągłej aktualizacji testów automatycznych. Jeśli wprowadzenie nowych lub zmiana dotychczasowych wymagań powoduje niezaliczenie pojedynczych testów lub całych zestawów testowych, nie należy pomijać niezaliczonych testów, tylko je skorygować.
- Planowanie wdrożenia. Należy zadbać o możliwość łatwego wdrażania, modyfikacji i ponownego wdrażania skryptów testowych.
- Wycofywanie testów w zależności od potrzeb. Należy zadbać o możliwość łatwego wycofywania z użycia skryptów testów automatycznych, które nie są już przydatne lub niezbędne.
- Monitorowanie i przywracanie sprawności systemu podlegającego testowaniu. Ciągłe uruchamianie przypadku testowego lub zbioru przypadków testowych wymaga w praktyce nieprzerwanego monitorowania systemu podlegającego testowaniu. Jeśli w systemie tym wystąpi błąd krytyczny (np. *crash*), struktura do testów automatycznych musi umożliwić przywrócenie sprawności systemu, pominięcie bieżącego przypadku testowego i wznowienie testowania od następnego przypadku.

Utrzymanie kodu związanego z automatyzacją testów może być skomplikowanym zadaniem. Zdarza się, że ilość kodu używanego do testowania dorównuje ilości kodu systemu podlegającego testowaniu, przez co kluczowego znaczenia nabiera utrzymywalność kodu używanego do testowania. Wynika to z faktu stosowania różnych narzędzi testowych i przeprowadzania różnego typu weryfikacji, a także z konieczności utrzymywania różnych testaliów (takich jak dane wejściowe do testów, wyrocznie testowe czy raporty z testów).

Mając na uwadze powyższe uwagi dotyczące utrzymania, oprócz ważnych elementów, które należy wykonać, warto unikać następujących działań:

- Nie należy tworzyć kodu uzależnionego od sposobu działania interfejsu (tj. kodu, na który miałyby wpływ zmiany w interfejsie graficznym lub w mniej istotnych obszarach API).
- Nie należy tworzyć takich mechanizmów automatyzacji testów, które są wrażliwe na zmiany danych lub w dużym stopniu uzależnione od określonych wartości danych (przykładem może być zależność danych wejściowych do testów od danych wyjściowych z innych testów).
- Nie należy tworzyć takiego środowiska automatyzacji, które jest wrażliwe na kontekst (np. na ustawienia daty i godziny w systemie operacyjnym, parametry lokalizacji systemu operacyjnego bądź zawartość innej aplikacji). W takim przypadku lepszym rozwiązaniem jest użycie w razie potrzeby zaślepek testowych do sterowania parametrami środowiska.

Im więcej kryteriów sukcesu zostanie spełnionych, tym wyższe będzie prawdopodobieństwo powodzenia projektu automatyzacji. Należy jednak pamiętać, że nie każde kryterium jest obowiązkowe, a w praktyce spełnienie wszystkich kryteriów jest rzadko spotykane. Dlatego przed rozpoczęciem projektu automatyzacji testów należy przeanalizować szanse jego powodzenia z punktu widzenia spełnionych i niespełnionych kryteriów, z uwzględnieniem ryzyk związanych z wybranym podejściem oraz kontekstu projektu. Po opracowaniu architektury automatyzacji testów (TAA) należy również zbadać, jakich elementów brakuje lub jakie elementy wymagają dalszego dopracowania.

2. Przygotowanie do automatyzacji testów — 165 min.

Słowa kluczowe

testowalność, sterownik, poziom inwazyjności, zaślepka, narzędzie do wykonywania testów, zaczep testowy, kierownik automatyzacji testów

Cele nauczania — przygotowanie do automatyzacji testów

2.1 Czynniki związane z systemem podlegającym testowaniu (SUT) wpływające na automatyzację testów

ALTA-E-2.1.1 (K4) Kandydat potrafi przeanalizować system podlegający testowaniu w celu określenia właściwego rozwiązania do automatyzacji.

2.2 Ocena i wybór narzędzi

ALTA-E-2.2.1 (K4) Kandydat potrafi przeanalizować narzędzia do automatyzacji testów w kontekście danego projektu i przedstawić wnioski/zalecenia techniczne.

2.3 Projektowanie pod kątem testowalności i automatyzacji

ALTA-E-2.3.1 (K2) Kandydat zna metody „projektowania pod kątem testowalności” i „projektowania pod kątem automatyzacji testów” mające zastosowanie do systemu podlegającego testowaniu (SUT).

2.1 Czynniki związane z systemem podlegającym testowaniu wpływające na automatyzację testów

Analizując kontekst systemu podlegającego testowaniu i jego środowiska należy zidentyfikować czynniki wpływające na automatyzację testów, a następnie wskazać na ich podstawie właściwe rozwiązanie. Wśród istotnych czynników można wymienić:

- Interfejsy systemu podlegającego testowaniu
Zautomatyzowane przypadki testowe wywołują określone akcje w systemie podlegającym testowaniu, w związku z czym system ten musi udostępniać interfejsy, za pośrednictwem których może być sterowany. Sterowanie może odbywać się przy użyciu elementów interfejsu użytkownika lub przy użyciu interfejsów oprogramowania niższego poziomu. Niektóre przypadki testowe mogą również korzystać z interfejsów na poziomie komunikacji (np. protokołu TCP/IP, magistrali USB lub autorskich interfejsów do przesyłania komunikatów).

Dekompozycja systemu podlegającego testowaniu umożliwia mechanizmom automatyzacji testów komunikowanie się z nim na różnych poziomach testów. Dzięki temu można zautomatyzować testy na określonym poziomie (np. na poziomie modułów i na poziomie systemu), o ile system podlegający testowaniu wspiera tego typu rozwiązania. W systemie może na przykład brakować interfejsu użytkownika, który umożliwiałby testowanie na poziomie modułów. W tej sytuacji muszą być dostępne inne, potencjalnie dostosowane do konkretnych potrzeb, interfejsy oprogramowania (zwane także zaczepami testowymi, ang. *test hooks*).

- Oprogramowanie innych firm
Systemy podlegające testowaniu często składają się zarówno z oprogramowania stworzonego przez organizację macierzystą, jak i oprogramowania dostarczonego przez inne firmy. W pewnych sytuacjach może być konieczne przetestowanie oprogramowania innej firmy, przy czym do ewentualnej automatyzacji tego procesu może być niezbędne inne rozwiązanie dotyczące automatyzacji testów — na przykład wykorzystujące interfejs API.
- Poziomy inwazyjności
Poszczególne podejścia do automatyzacji testów (korzystające z różnych narzędzi) różnią się poziomem inwazyjności. Im większa jest liczba zmian w systemie podlegającym testowaniu spowodowana testowaniem automatycznym, tym wyższy jest poziom inwazyjności. Korzystanie z wyspecjalizowanych interfejsów oprogramowania charakteryzuje się wysokim poziomem inwazyjności — w przeciwieństwie do korzystania z istniejących elementów interfejsu użytkownika. W przypadku wykorzystania elementów sprzętowych systemu podlegającego testowaniu (takich jak klawiatury, przełączniki ręczne, ekrany dotykowe czy interfejsy komunikacyjne) mamy do czynienia z jeszcze wyższym poziomem inwazyjności.²

Problem z wyższymi poziomami inwazyjności polega na tym, że wiążą się one z większym ryzykiem wystąpienia fałszywych alarmów. W rezultacie, rozwiązanie dla testów automatycznych (TAS) może zgłaszać awarie wynikające z wysokiego poziomu inwazyjności narzuconego przez testy, które prawdopodobnie nie wystąpiłyby podczas korzystania z systemu oprogramowania w rzeczywistych warunkach eksploatacji. Z drugiej strony, testowanie z wysokim poziomem inwazyjności jest zwykle prostszym rozwiązaniem dla podejścia do automatyzacji testów.

² Wyjaśnienie autora, Armina Borna: gdy tworzone są w SUT interfejsy dedykowane do automatyzacji testów (a nie po to, by wspierać inne wymagania), wtedy poziom inwazyjności jest wysoki. Gdy interfejsy wbudowane w SUT używane są po to, żeby wspierać dowolne wymagania (także automatyzację testów), wtedy poziom inwazyjności jest niski. Podczas korzystania z SUT z już istniejących zewnętrznych systemów, poziom inwazyjności będzie bardzo niski. Zewnętrzne interfejsy mogą być elektronicznymi sygnałami (dla fizycznych przełączników), sygnałów USB dla urządzeń USB (takich jak klawiatura).

- Różne architektury systemów podlegających testowaniu
Różne architektury SUT mogą wymagać stosowania różnych rozwiązań do automatyzacji testów. Na przykład system napisany w języku C++ z wykorzystaniem technologii COM wymaga innego podejścia niż system napisany w języku Python. Nie wyklucza to zastosowania wspólnej strategii automatyzacji testów obejmującej różne architektury, ale musi to być strategia hybrydowa z możliwością ich wsparcia, uwzględniająca specyfikę poszczególnych środowisk.
- Wielkość i złożoność systemu podlegającego testowaniu
Należy wziąć pod uwagę wielkość i złożoność obecnej wersji systemu podlegającego testowaniu oraz plany dotyczące jego dalszego rozwoju. W przypadku niewielkiego, prostego systemu podlegającego testowaniu, złożone i bardzo elastyczne podejście do automatyzacji testów może okazać się nieuzasadnione — w takiej sytuacji lepiej jest zastosować proste rozwiązania. Z drugiej strony, niewskazane jest stosowanie prostego (nieskomplikowanego) podejścia w przypadku bardzo dużego i złożonego systemu. Zdarzają się oczywiście sytuacje, w których można z powodzeniem zacząć od prostych, zakrojonych na niewielką skalę, rozwiązań nawet w przypadku systemu o dużej złożoności, ale podejście takie powinno mieć wyłącznie charakter tymczasowy (więcej informacji na ten temat zawiera Rozdział 3.).

Niektóre z opisanych powyżej czynników (np. wielkość i złożoność systemu bądź dostępność interfejsów oprogramowania) można określić dopiero po udostępnieniu systemu podlegającego testowaniu, ale w większości przypadków prace nad automatyzacją testów powinny rozpocząć się zanim system ten zostanie udostępniony. Z jednej strony, wymaga to przyjęcia pewnych założeń szacunkowych, a z drugiej - stwarza inżynierowi automatyzacji testów (TAE) okazję do określenia niezbędnych interfejsów oprogramowania (więcej informacji na ten temat zawiera podrozdział 2.3).

Planowanie automatyzacji testów można rozpocząć jeszcze przed powstaniem systemu podlegającego testowaniu, na przykład:

- Gdy są już znane wymagania (funkcjonalne lub нефunkcjonalne), można na ich podstawie wybrać testy potencjalnie nadające się do zautomatyzowania i zidentyfikować sposoby ich przeprowadzania. Po wybraniu testów można rozpocząć planowanie automatyzacji, w tym identyfikowanie wymagań związanych z automatyzacją oraz określanie strategii automatyzacji testów.
- W fazie opracowywania architektury i projektu technicznego można przystąpić do projektowania interfejsów oprogramowania wspomagających testowanie.

2.2 Ocena i wybór narzędzi

Główna odpowiedzialność za proces wyboru i oceny narzędzi spoczywa na kierowniku automatyzacji testów (TAM), jednak inżynier automatyzacji testów (TAE) również będzie zaangażowany w ten proces, udzielając TAM niezbędnych informacji oraz wykonując cały szereg czynności związanych z oceną i wyborem narzędzi. Koncepcja procesu oceny i wyboru narzędzi została wprowadzona w sylabusie poziomu podstawowego, a szczegółowe informacje na temat tego procesu zawiera sylabus poziomu zaawansowanego - Kierownik Testów [ISTQB-AL-TM].

TAE uczestniczy we wszystkich etapach procesu oceny i wyboru narzędzi, a zwłaszcza w:

- ocenie dojrzałości organizacyjnej i identyfikacji możliwości wsparcia przez narzędzia testowe,
- ocenie właściwych celów, jakie należy osiągnąć w zakresie wsparcia przez narzędzia testowe,
- określeniu potencjalnie przydatnych narzędzi i zbieraniu informacji na ich temat,
- analizowaniu informacji o narzędziach z punktu widzenia celów i ograniczeń projektu,
- szacowaniu stosunku kosztów do przewidywanych korzyści na podstawie rzetelnego uzasadnienia biznesowego,
- rekomendowaniu właściwego narzędzia,
- określaniu kompatybilności narzędzia z komponentami systemu podlegającego testowaniu.

Narzędzia do automatyzacji testów funkcjonalnych często nie spełniają wszystkich oczekiwań i nie uwzględniają wszystkich sytuacji, które mogą wystąpić w ramach projektu automatyzacji. Poniżej przedstawiono kilka przykładów tego rodzaju problemów (oczywiście poniższa lista nie jest wyczerpująca):

Sytuacja	Przykłady	Potencjalne rozwiązania
Interfejs narzędzia nie współpracuje z innymi wdrożonymi wcześniej narzędziami.	<ul style="list-style-type: none"> W wyniku aktualizacji narzędzia do zarządzania testami zmienił się interfejs łączący. Informacje udzielone przez dział obsługi sprzedażowej okazały się błędne, przez co nie wszystkie dane można było przenieść do narzędzia raportującego. 	<ul style="list-style-type: none"> Przed każdą aktualizacją należy uważnie zapoznać się z opisem wydania (ang. <i>release notes</i>), a w przypadku dużych migracji należy przetestować produkt przed przekazaniem go do środowiska produkcyjnego. Należy poprosić o przeprowadzenie demonstracji narzędzia w miejscu instalacji, z wykorzystaniem rzeczywistego systemu podlegającego testowaniu. Należy zwrócić się o pomoc do dostawcy i/lub skorzystać z forów dyskusyjnych dla użytkowników.
Niektóre zależności systemu podlegającego testowaniu uległy zmianie i nie są już obsługiwane przez narzędzie testowe.	<ul style="list-style-type: none"> Dział rozwoju zaktualizował oprogramowanie do najnowszej wersji środowiska Java. 	<ul style="list-style-type: none"> Należy zsynchronizować aktualizacje środowiska programistycznego i testowego z aktualizacjami narzędzia do automatyzacji testów.
Nie można przechwycić obiektu w graficznym interfejsie użytkownika (GUI).	<ul style="list-style-type: none"> Obiekt jest widoczny, ale narzędzie do automatyzacji testów nie może wejść z nim w interakcję. 	<ul style="list-style-type: none"> Programiści powinni spróbować zastosować wyłącznie dobrze znane technologie i obiekty. Przed zakupem narzędzia do automatyzacji testów należy przeprowadzić projekt pilotażowy. Należy zlecić programistom określenie standardów dotyczących obiektów.
Narzędzie sprawia wrażenie bardzo skomplikowanego.	<ul style="list-style-type: none"> Narzędzie ma bardzo bogaty zestaw funkcji, jednak tylko część z nich będzie faktycznie wykorzystywana. 	<ul style="list-style-type: none"> Należy poszukać sposobu na ograniczenie dostępnego zestawu funkcji poprzez usunięcie zbędnych elementów z paska narzędzi. Należy wybrać licencję dostosowaną do rzeczywistych potrzeb. Należy poszukać narzędzi lepiej dopasowanych do rzeczywistych potrzeb związanych z wymaganą funkcjonalnością.
Konflikt z innymi systemami	<ul style="list-style-type: none"> Narzędzie do automatyzacji testów przestaje działać po zainstalowaniu innego oprogramowania (lub odwrotnie). 	<ul style="list-style-type: none"> Przed rozpoczęciem instalacji należy zapoznać się z opisem wydania lub wymaganiami technicznymi. Należy uzyskać od dostawcy potwierdzenie, że narzędzie nie będzie wpływać na działanie innych narzędzi. Należy zadać pytanie na forach dyskusyjnych dla użytkowników.
Wpływ na system podlegający testowaniu	<ul style="list-style-type: none"> System podlegający testowaniu reaguje inaczej podczas używania narzędzia do automatyzacji testów lub po jego użyciu (np. wydłużeniu czasu odpowiedzi). 	<ul style="list-style-type: none"> Należy używać narzędzia, które nie wymaga wprowadzania zmian w systemie podlegającym testowaniu (instalowania bibliotek itp.).
Dostęp do kodu	<ul style="list-style-type: none"> Narzędzie do automatyzacji testów zmienia fragmenty kodu 	<ul style="list-style-type: none"> Należy używać narzędzia, które nie wymaga ingerencji w kod

	źródłowego.	źródłowy (instalowania bibliotek itp.).
Ograniczone zasoby (głównie w środowiskach wbudowanych)	<ul style="list-style-type: none"> W środowisku testowym dostępna jest ograniczona ilość wolnych zasobów lub brakuje już zasobów (np. pamięci). 	<ul style="list-style-type: none"> Należy zapoznać się z opisem wydania i skonsultować środowisko z dostawcą narzędzia, aby upewnić się, że nie wystąpią problemy. Należy zadać pytanie na forach dyskusyjnych dla użytkowników.
Aktualizacje	<ul style="list-style-type: none"> Aktualizacja nie powoduje przeniesienia części danych bądź powoduje uszkodzenie istniejących skryptów testów automatycznych, danych lub konfiguracji. Aktualizacja wymaga innego (lepszego) środowiska. 	<ul style="list-style-type: none"> Należy przetestować aktualizację w środowisku testowym i uzyskać od dostawcy potwierdzenie, że migracja przebiegnie prawidłowo. Należy zapoznać się z wymaganiami wstępnymi i zdecydować, czy korzyści wynikające z aktualizacji są warte związanego z nią nakładu pracy. Należy szukać wsparcia na forach dyskusyjnych dla użytkowników.
Zabezpieczenia	<ul style="list-style-type: none"> Narzędzie do automatyzacji testów wymaga informacji niedostępnych dla inżyniera automatyzacji testów. 	<ul style="list-style-type: none"> Należy przyznać inżynierowi automatyzacji testów odpowiednie uprawnienia dostępu.
Niezgodność różnych środowisk i platform	<ul style="list-style-type: none"> Automatyzacja testów nie działa prawidłowo we wszystkich środowiskach lub na wszystkich platformach. 	<ul style="list-style-type: none"> Należy zaimplementować testy automatyczne w sposób jak najbardziej niezależny od konkretnych narzędzi, co pozwoli ograniczyć koszty związane z używaniem różnych narzędzi.

2.3 Projektowanie pod kątem testowalności i automatyzacji

Równolegle do projektowania i implementacji innych funkcji systemu podlegającego testowaniu należy również zadbać o jego testowalność (czyli dostępność interfejsów oprogramowania wspomagających testowanie, na przykład umożliwiających kontrolę systemu i obserwowanie jego zachowania). Może to zrobić architekt oprogramowania (testowalność jest jednym z wielu wymagań niefunkcjonalnych systemu), ale często jest to zadanie wykonywane przez inżyniera automatyzacji testów lub przy jego udziale.

Projektowanie pod kątem testowalności można podzielić na kilka obszarów:

- **Obserwowalność.** System podlegający testowaniu musi udostępniać interfejsy umożliwiające wgląd w jego działanie. Na przykład, przypadki testowe mogą za ich pomocą sprawdzać, czy rzeczywiste zachowanie systemu jest zgodne z zachowaniem oczekiwanym.
- **Kontrolowalność.** System podlegający testowaniu musi udostępniać interfejsy, za pomocą których można w nim wykonywać określone działania. Mogą to być elementy interfejsu użytkownika, wywołania funkcji, elementy komunikacyjne (np. protokół TCP/IP lub USB), sygnały elektroniczne (dla przełączników fizycznych) itp.
- **Jednoznacznie zdefiniowana architektura.** Trzecim istotnym elementem projektowania pod kątem testowalności jest architektura oferująca wyraźnie określone i zrozumiałe interfejsy, umożliwiające kontrolowanie i zapewniające widoczność na wszystkich poziomach testów.

Inżynier automatyzacji testów bierze pod uwagę różne sposoby testowania (w tym testowanie automatyczne) pozwalające skutecznie przetestować system podlegający testowaniu (co oznacza przetestowanie właściwych obszarów i wykrycie krytycznych usterek) oraz efektywnie (czyli bez nadmiernego nakładu pracy). Wszędzie tam, gdzie konieczne jest użycie określonych interfejsów oprogramowania, interfejsy takie muszą zostać określone przez TAE i zaimplementowane przez programistę. Wymagania dotyczące testowalności

oraz ewentualnych dodatkowych interfejsów oprogramowania należy określić na wczesnym etapie projektu, aby można było zaplanować i zabudżetować prace programistyczne.

Poniżej podano kilka przykładów zastosowania interfejsów oprogramowania wspomagających testowanie:

- Wykorzystanie zaawansowanych funkcji tworzenia skryptów oferowane przez nowoczesne arkusze kalkulacyjne.
- Zastosowanie zaślepek lub makiet (ang. *mocks*) do symulowania pracy oprogramowania i/lub sprzętu (np. elektronicznych transakcji finansowych, usług oprogramowania, wyspecjalizowanych serwerów, układów elektronicznych czy elementów mechanicznych), które nie są jeszcze dostępne lub są zbyt kosztowne. Metoda ta pozwala przetestować oprogramowanie pomimo braku określonego interfejsu.
- Testowanie warunków wystąpienia błędów za pomocą interfejsów oprogramowania (lub zaślepek i sterowników). Metodę tę można zastosować na przykład w przypadku urządzenia wyposażonego w wewnętrzny dysk twardy (HDD). Oprogramowanie sterujące dyskiem (tj. sterownik) musi zostać przetestowane pod kątem zachowania w przypadku awarii lub zużycia napędu. Oczekiwanie na faktyczną awarię dysku jest rozwiązaniem nieefektywnym i daje mało wiarygodne wyniki. W tej sytuacji można zaimplementować interfejsy oprogramowania symulujące działanie uszkodzonego lub wolniej pracującego dysku, które pozwalają sprawdzić, czy oprogramowanie sterownika funkcjonuje prawidłowo (tj. generuje komunikaty o błędach, ponawia próby wykonania operacji itd.).
- Wykorzystanie alternatywnych interfejsów oprogramowania do testowania SUT przed udostępnieniem interfejsu użytkownika (podejście takie jest często uznawane za bardziej korzystne). Metodę tę można zastosować na przykład do oprogramowania wbudowanego działającego w systemach technicznych, które musi monitorować temperaturę wewnątrz urządzenia i uruchamiać funkcję chłodzenia, gdy temperatura przekroczy określoną wartość. Użycie interfejsu oprogramowania dostarczającego informacji o temperaturze pozwala przetestować funkcję bez korzystania ze sprzętu.
- Ocena zachowania systemu podlegającego testowaniu w różnych stanach na podstawie testowania przejść pomiędzy stanami. Do sprawdzania, czy stan SUT znajduje się w prawidłowym stanie, można użyć specjalnie w tym celu zaprojektowanego interfejsu oprogramowania (choć wiąże się to z pewnym ryzykiem — patrz informacje o poziomie inwazyjności w podrozdziale 2.1).

W ramach projektowania pod kątem automatyzacji należy wziąć pod uwagę następujące zalecenia:

- Kompatybilność z dotychczasowymi narzędziami testowymi powinna być określona na wczesnym etapie prac.
- Kompatybilność narzędzi testowych ma kluczowe znaczenie, ponieważ może wpływać na możliwość zautomatyzowania testów dotyczących ważnej funkcjonalności (np. niekompatybilność z elementem sterującym siecią uniemożliwia przeprowadzenie wszystkich testów korzystających z tego elementu).
- Stworzenie rozwiązania może wymagać utworzenia kodu programu i wywołań interfejsów API.

Projektowanie pod kątem testowalności jest niezwykle ważnym elementem dobrego podejścia do automatyzacji testów, a także może również przynieść korzyści w obszarze testowania manualnego.

3. Ogólna architektura testów automatycznych — 270 min.

Słowa kluczowe

podejście „rejestrowanie/odtworzenie”, testowanie sterowane danymi, ogólna architektura testów automatycznych, testowanie oparte na słowach kluczowych, technika skryptów liniowych, testowanie oparte na modelu, technika skryptów sterowanych przez proces, technika skryptów ustrukturalizowanych, warstwa adaptacji testów, architektura testów automatycznych, struktura do testów automatycznych, rozwiązanie dla testów automatycznych, warstwa definicji testów, warstwa wykonywania testów, warstwa generowania testów

Cele nauczania — ogólna architektura testów automatycznych

3.1 Wprowadzenie do ogólnej architektury testów automatycznych (gTAA)

ALTA-E-3.1.1 (K2) Kandydat potrafi objaśnić strukturę ogólnej architektury testów automatycznych (gTAA).

3.2 Projektowanie architektury automatyzacji testów (TAA)

ALTA-E-3.2.1 (K4) Kandydat potrafi zaprojektować odpowiednią architekturę automatyzacji testów (TAA) na potrzeby danego projektu.

ALTA-E-3.2.2 (K2) Kandydat potrafi wyjaśnić, jaką rolę odgrywają poszczególne warstwy architektury automatyzacji testów.

ALTA-E-3.2.3 (K2) Kandydat zna uwarunkowania związane z projektowaniem architektury automatyzacji testów.

ALTA-E-3.2.4 (K4) Kandydat potrafi przeanalizować wymagania związane z implementacją, używaniem i utrzymaniem wymagań danego rozwiązania dla testów automatycznych (TAS).

3.3 Tworzenie rozwiązania dla testów automatycznych (TAS)

ALTA-E-3.3.1 (K3) Kandydat potrafi wykorzystać komponenty ogólnej architektury testów automatycznych (gTAA) do skonstruowania architektury testów automatycznych (TAA) przeznaczonej do określonych zastosowań.

ALTA-E-3.3.2 (K2) Kandydat potrafi wyjaśnić, jakie czynniki należy wziąć pod uwagę przy identyfikowaniu możliwości ponownego wykorzystania komponentów.

3.1 Wprowadzenie do ogólnej architektury testów automatycznych (gTAA)

Rolą inżyniera automatyzacji testów jest projektowanie, rozwijanie, implementowanie i utrzymanie rozwiązań dla testów automatycznych (TAS). Podczas opracowywania każdego rozwiązania należy wykonać podobne zadania, odpowiedzieć na podobne pytania oraz rozwiązać i spriorytetyzować podobne zagadnienia. Te powtarzające się koncepcje, kroki i podejścia do automatyzacji testów są podstawą do opracowania ogólnej architektury testów automatycznych (ang. *generic Test Automation Architecture* - gTAA).

gTAA przedstawia warstwy, komponenty i interfejsy gTAA, które są następnie redefiniowane w celu stworzenia konkretnej architektury automatyzacji testów (TAA) na potrzeby konkretnego rozwiązania dla testów automatycznych. Architektura taka pozwala zbudować wymagane rozwiązanie w sposób ustrukturyzowany i modułowy poprzez:

- Definiowanie pojęć, warstw, usług i interfejsów TAS w celu umożliwienia realizacji rozwiązań dla testów automatycznych (TAS) w oparciu zarówno o komponenty opracowane we własnym zakresie, jak i przez podmioty zewnętrzne.
- Zapewnienie uproszczonych komponentów umożliwiających skuteczne i efektywne tworzenie mechanizmów automatyzacji testów.
- Ponowne wykorzystanie komponentów związanych z automatyzacją testów do tworzenia innych lub rozwijania istniejących TAS na potrzeby różnych linii asortymentu produktów programowych oraz technologii i narzędzi programistycznych.
- Ułatwienie pielęgnacji i rozwoju TAS.
- Definiowanie podstawowych funkcji dla użytkownika TAS.

W skład rozwiązania do automatyzacji testów wchodzi zarówno środowisko testowe (wraz z jego artefaktami), jak i zestawy testów (tj. zbiory przypadków testowych wraz z danymi testowymi). Do realizacji TAS można wykorzystać strukturę do testów automatycznych (ang. *Test Automation Framework* - TAF), która ułatwia stworzenie środowiska testowego oraz zapewnia niezbędne narzędzia, jarzma testowe i biblioteki pomocnicze.

Zaleca się, aby architektura testów automatycznych (TAA) dla rozwiązania do automatyzacji testów była zgodna z poniższymi zasadami, których przestrzeganie ułatwia tworzenie, rozwijanie i utrzymanie tego typu rozwiązań:

- Pojedyncza odpowiedzialność (jednozadaniowość). Każdy komponent TAS musi mieć pojedynczą odpowiedzialność (realizować jeden cel), a ta odpowiedzialność musi być w całości zamknięta w komponencie. Innymi słowy, każdy komponent rozwiązania dla testów automatycznych powinien odpowiadać za wykonywanie dokładnie jednej czynności, na przykład za generowanie słów kluczowych lub danych, tworzenie scenariuszy testowych, wykonywanie przypadków testowych, rejestrowanie rezultatów bądź generowanie raportów z wykonania testów.
- Rozszerzalność (patrz np. zasada „otwarte-zamknięte” wg B. Meyera). Każdy komponent TAS musi być otwarty na rozszerzenia, ale zamknięty na modyfikacje. Zgodnie z tą zasadą, powinna istnieć możliwość modyfikowania lub wzbogacania zachowania komponentów, ale tylko pod warunkiem, że nie wpływa to negatywnie na kompatybilność wsteczną funkcjonalności.
- Zastępowalność (patrz np. zasada podstawienia wg B. Liskov – ang. *Liskov substitution principle*). Musi istnieć możliwość wymiany każdego komponentu TAS na inny, bez wywierania wpływu na ogólne zachowanie całego rozwiązania dla testów automatycznych. Oznacza to, że po zastąpieniu danego komponentu jednym lub kilkoma innymi komponentami zachowanie rozwiązania musi pozostać bez zmian.
- Segregacja komponentów (patrz np. zasada rozdzielania interfejsów wg R. C. Martina). Zastosowanie kilku komponentów specjalizowanych jest lepsze niż zastosowanie jednego komponentu ogólnego, wielofunkcyjnego, a wyeliminowanie zbędnych współzależności ułatwia wymianę i utrzymanie komponentów.
- Odwracanie zależności. Działanie komponentów TAS musi opierać się na abstrakcjach, a nie na szczegółach niskiego poziomu. Innymi słowy, funkcjonowanie komponentów nie powinno być uzależnione od konkretnych scenariuszy testów automatycznych.

Rozwiązanie dla testów automatycznych oparte na ogólnej architekturze testów automatycznych jest zwykle implementowane przy użyciu zbioru narzędzi (wraz z wtyczkami) i/lub komponentów. Warto przy tym zauważyć, że gTAA jest neutralna dla dostawcy, czyli nie określa z góry żadnej konkretnej metody,

technologii ani narzędzi służących do realizacji TAS. Architektura tę (gTAA) można zaimplementować przy użyciu dowolnego podejścia do inżynierii oprogramowania (np. metodą strukturalną, obiektową, zorientowaną na usługi lub sterowaną modelem) oraz z wykorzystaniem dowolnych technologii i narzędzi programistycznych. W praktyce rozwiązania do automatyzacji testów często wdraża się z wykorzystaniem gotowych narzędzi, chociaż zwykle konieczne jest także dodanie określonych funkcji i/lub dostosowanie narzędzi do potrzeb systemu podlegającego testowaniu.

Wśród wytycznych i modeli wzorcowych związanych z tworzeniem rozwiązań do automatyzacji testów należy również wymienić standardy w dziedzinie inżynierii oprogramowania dotyczące wybranego cyklu wytwarzania oprogramowania (ang. *Software Development Lifecycle* - SDLC), technologie programowania, standardy formatowania itd. Nauczanie ogólnych zasad inżynierii oprogramowania nie jest przedmiotem niniejszego sylabusu, ale należy zaznaczyć, że inżynier automatyzacji testów powinien wykazywać się umiejętnościami, doświadczeniem i specjalistyczną wiedzą w tej dziedzinie.

Ponadto TAE musi znać standardy branżowe i najlepsze praktyki dotyczące tworzenia kodu i dokumentacji oraz korzystać z nich podczas tworzenia TAS, co pozwoli zwiększyć utrzymywalność, niezawodność i zabezpieczenia tego rozwiązania. Powyższe standardy dotyczą zwykle konkretnych dziedzin. Można wśród nich wymienić:

- standard MISRA dotyczący języków C i C++,
- standard JSF dotyczący tworzenia kodu w języku C++,
- reguły AUTOSAR dotyczące środowiska MathWorks Matlab/Simulink®.

3.1.1 Omówienie ogólnej architektury testów automatycznych (gTAA)

Ogólna architektura testów automatycznych (ang. *generic Test Automation Architecture* - gTAA) dzieli się na kilka poziomych warstw, które odpowiadają za:

- generowanie testów,
- definiowanie testów,
- wykonywanie testów,
- adaptację testów.

W skład gTAA (patrz Rys. 1. — Ogólna architektura testów automatycznych) wchodzi następujące elementy:

- Warstwa generowania testów, która umożliwia manualne lub automatyczne projektowanie przypadków testowych. Warstwa ta udostępnia mechanizmy służące do projektowania przypadków testowych.
- Warstwa definiowania testów, która umożliwia definiowanie i implementowanie zestawów testowych i/lub przypadków testowych. Warstwa ta pozwala oddzielić proces definiowania testów od technologii i narzędzi systemu podlegającego testowaniu i/lub systemu testowego. Ponadto zawiera mechanizmy umożliwiające definiowanie testów wysokiego i niskiego poziomu, które są realizowane w ramach danych testowych, przypadków testowych, procedur testowych i komponentów bibliotek testowych (lub ich kombinacji).
- Warstwa wykonywania testów, która wspomaga wykonywanie przypadków testowych i logowanie testu. Warstwa ta udostępnia narzędzie do automatycznego wykonywania wybranych testów oraz komponent rejestrująco-raportujący.
- Warstwa adaptacji testów, w ramach której dostępny jest kod niezbędny do dostosowywania testów automatycznych do potrzeb poszczególnych komponentów lub interfejsów systemu podlegającego testowaniu. Warstwa ta oferuje różne adaptery umożliwiające łączenie się z SUT za pośrednictwem API, protokołów, usług itd.
- Interfejsy do zarządzania projektami, konfiguracją i testami w odniesieniu do automatyzacji testów. Przykładem może być interfejs między zarządzaniem testami a warstwą adaptacji testów, który odpowiada za wybór i konfigurowanie odpowiednich adapterów w zależności od wybranej konfiguracji testowej.

Interfejsy między warstwami gTAA oraz ich komponentami są zwykle specyficzne dla danego projektu, w związku z czym nie będą szerzej omawiane w tym dokumencie.

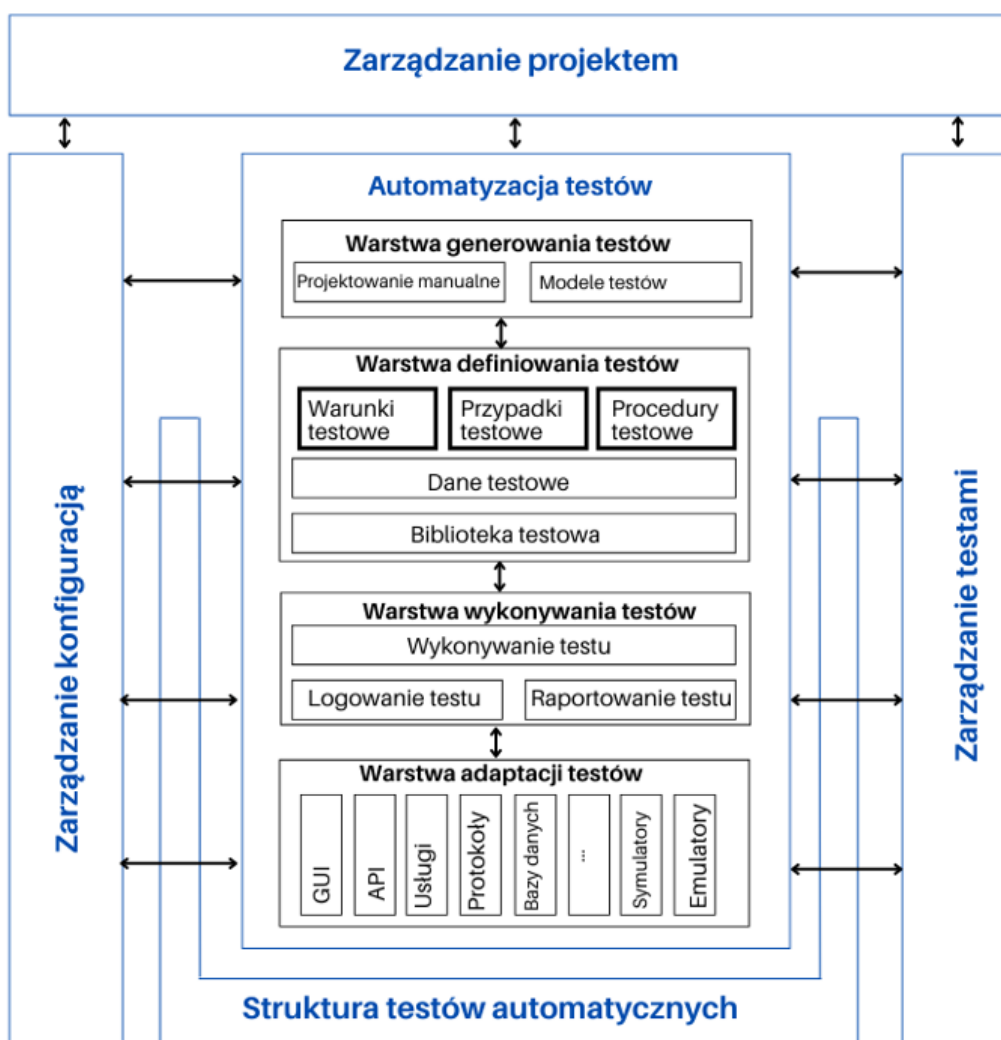
Należy podkreślić, że powyższe warstwy mogą, ale nie muszą, występować w konkretnym TAS. Na przykład:

- Automatyzacja wykonywania testów wymaga zastosowania warstwy wykonywania testów i warstwy adaptacji testów. Warstwy te nie muszą być rozdzielone, co oznacza, że można je zrealizować łącznie, na przykład w ramach struktur do testów modułowych.

- Automatykacja procesu definiowania testów wymaga zastosowania warstwy definiowania testów.
- Automatykacja procesu generowania testów wymaga zastosowania warstwy generowania testów.

Implementacja TAS przebiega najczęściej od dołu do góry, ale przydatne bywają również inne podejścia, takie jak automatyczne generowanie testów dla testów manualnych. Generalnie zaleca się implementowanie TAS metodą przyrostową (np. w formie sprintów), ponieważ pozwala to najszybciej skorzystać z tego rozwiązania i potwierdzić generowaną przez nie wartość dodaną. Ponadto wskazane jest, by w projekcie automatyzacji testów uwzględnić fazę weryfikacji koncepcji (ang. *proof of concept*).

Każdy projekt automatyzacji testów musi być rozumiany, skonfigurowany i zarządzany jako projekt rozwoju oprogramowania i wymaga dedykowanego zarządzania projektem. Można przy tym oddzielić zarządzanie projektem rozwoju TAF (której zadaniem jest automatyzacja testów na poziomie całego przedsiębiorstwa, rodzin produktów lub linii produktów) od zarządzania projektem TAS (którego zadaniem jest automatyzacja testów na poziomie konkretnego produktu).



Rysunek 1. Ogólna architektura testów automatycznych

3.1.2 Warstwa generowania testów

Warstwa generowania testów składa się z narzędzi umożliwiających:

- manualne projektowanie przypadków testowych,
- opracowywanie, rejestrowanie i wyprowadzanie danych testowych,
- automatyczne generowanie przypadków testowych na podstawie modeli definiujących system podlegający testowaniu i/lub jego środowisko (tj. zautomatyzowane testowanie oparte na modelu).

Komponenty wchodzące w skład tej warstwy służą do:

- edytowania struktur zestawów testowych i poruszania się po nich,
- tworzenia powiązań pomiędzy przypadkami testowymi a celami testów lub wymaganiami zdefiniowanymi dla systemu podlegającego testowaniu,
- dokumentowania projektu testów.

W przypadku automatycznego generowania testów można również uwzględnić możliwość:

- modelowania systemu podlegającego testowaniu, jego środowiska i/lub systemu testowego,
- definiowania dyrektyw testowych oraz konfigurowania/parametryzowania algorytmów generowania testów,
- śledzenia powiązań pomiędzy wygenerowanymi testami a modelem (lub jego elementami).

3.1.3 Warstwa definiowania testów

Warstwa definiowania testów składa się z narzędzi umożliwiających:

- określenie przypadków testowych (wysokiego i/lub niskiego poziomu),
- definiowanie danych testowych na potrzeby przypadków testowych niskiego poziomu,
- określenie procedur testowych na potrzeby przypadku testowego lub zbioru przypadków testowych,
- definiowanie skryptów testowych służących do wykonywania przypadków testowych,
- zapewnienie dostępu do niezbędnych bibliotek testowych (np. w przypadku podejść opartych na słowach kluczowych).

Komponenty wchodzące w skład tej warstwy służą do:

- partycjonowania/ograniczania, parametryzowania lub tworzenia instancji danych testowych,
- określenia sekwencji testowych lub kompletnych zachowań testowych (łącznie z instrukcjami sterującymi i wyrażeniami), aby je sparametryzować i/lub pogrupować,
- dokumentowania danych testowych, przypadków testowych i/lub procedur testowych.

3.1.4 Warstwa wykonywania testów

Warstwa wykonywania testów składa się z narzędzi umożliwiających:

- automatyczne wykonywanie przypadków testowych,
- logowanie wykonywania przypadków testowych,
- raportowanie rezultatów testów.

Warstwa wykonywania testów może składać się z komponentów umożliwiających:

- ustanowienie (ang. *set up*) i rozmontowanie (ang. *tear down*) konfiguracji SUT dla wykonania testów,
- ustanowienie i rozmontowanie konfiguracji zestawów testowych (tj. zbiorów przypadków testowych wraz z danymi testowymi),
- konfigurowanie i parametryzowanie konfiguracji testowej,
- interpretowanie danych testowych i przypadków testowych oraz przekształcanie ich w wykonywalne skrypty,
- zaadaptowanie systemu testowego i/lub systemu podlegającego testowaniu (SUT) w celu logowania danych związanych z wykonywaniem testów (z funkcją filtrowania) oraz/lub wstrzykiwania błędów,
- analizowanie odpowiedzi systemu podlegającego testowaniu podczas wykonywania testów w celu pokierowania kolejnymi przebiegami testów,
- dokonywanie walidacji odpowiedzi systemu podlegającego testowaniu (tj. porównywanie rzeczywistych rezultatów z oczekiwanymi) w przypadku wykonywania zautomatyzowanych przypadków testowych,
- nadzór nad wykonywaniem testów automatycznych w czasie.

3.1.5 Warstwa adaptacji testów

Warstwa adaptacji testów składa się z narzędzi umożliwiających:

- sterowanie jarzmem testowym,
- interakcję z systemem podlegającym testowaniu,
- monitorowanie systemu podlegającego testowaniu,
- symulację lub emulację środowiska systemu podlegającego testowaniu.

Warstwa adaptacji testów oferuje funkcje umożliwiające:

- mediację między neutralnymi technologicznie definicjami testów a konkretnymi wymaganiami technicznymi SUT i urządzeń testowych,
- stosowanie różnych adapterów specyficznych dla technologii do interakcji z SUT,
- wykonywanie testów z wykorzystaniem wielu urządzeń/interfejsów testowych bądź wykonywanie testów lokalnie.

3.1.6 Zarządzanie konfiguracją TAS

W ramach prac nad rozwiązaniem dla testów automatycznych powstają zwykle różne iteracje/wersje, które muszą być zgodne z odpowiednimi iteracjami/wersjami systemu podlegającego testowaniu. Zarządzanie konfiguracją TAS może uwzględniać między innymi:

- modele testów,
- definicje/specyfikacje testów obejmujące dane testowe, przypadki testowe i biblioteki,
- skrypty testowe,
- mechanizmy wykonywania testów oraz narzędzia i komponenty uzupełniające,
- adaptory testowe dla systemu podlegającego testowaniu,
- symulatory i emulatory dla środowiska systemu podlegającego testowaniu,
- rezultaty testów i raporty z testów.

Powyższe elementy stanowią testalia i muszą być dostępne we właściwej wersji, zgodnej z wersją systemu podlegającego testowaniu. Ponadto, w pewnych sytuacjach może być konieczne przywrócenie poprzednich wersji TAS, na przykład jeśli konieczne jest odtworzenie defektów polowych (ang. *field issues*) występujących w starszych wersjach systemu podlegającego testowaniu. Możliwość taką zapewnia odpowiednie zarządzanie konfiguracją.

3.1.7 Zarządzanie projektem w zakresie TAS

Każdy projekt automatyzacji testów jest projektem programistycznym, w związku z czym należy nim zarządzać tak samo, jak każdym innym projektem tego typu. W trakcie prac nad TAS inżynier automatyzacji testów musi wykonywać zadania przypisane do poszczególnych faz przyjętego cyklu wytwarzania oprogramowania (SDLC). Ponadto musi on zdawać sobie sprawę, że środowisko programistyczne, w którym powstaje TAS, powinno być zaprojektowane w sposób umożliwiający łatwe pozyskiwanie informacji o statusie (metryk) lub automatyczne raportowanie ich osobom kierującym projektem związanym z TAS.

3.1.8 Obsługa zarządzania testami na poziomie TAS

Rozwiązanie dla testów automatycznych musi wspierać zarządzanie testami wykonywanymi w systemie podlegającym testowaniu. Raporty z testów, włączając w to dzienniki testów i rezultaty testów, muszą być łatwo wyodrębnione bądź automatycznie dostarczane do kierownictwa testów (osób lub systemu) dla SUT.

3.2 Projektowanie architektury automatyzacji testów (TAA)

3.2.1 Wprowadzenie do projektowania TAA

Aby zaprojektować architekturę automatyzacji testów, należy wykonać kilka istotnych czynności, które można uszeregować stosownie do potrzeb projektu automatyzacji lub potrzeb całej organizacji. Czynności te omówiono w kolejnych punktach. Faktyczna liczba wykonywanych czynności (mniejsza lub większa ich liczba) zależy od złożoności rozwiązania dla testów automatycznych.

Zarejestrowanie wymagań niezbędnych do zdefiniowania odpowiedniej TAA

Aby określić wymagania dotyczące podejścia do automatyzacji testów, należy uzyskać odpowiedź na następujące pytania:

- Którą czynność lub fazę procesu testowego należy zautomatyzować (np. zarządzanie testami, projektowanie testów, generowanie testów czy wykonywanie testów)? Warto przy tym zaznaczyć, że automatyzacja testów wiąże się również z udoskonaleniem zasadniczego procesu testowego poprzez umieszczenie etapu generowania testów między etapami projektowania i implementacji testów.
- Jaki poziom testów powinien być wspierany (np. poziom modułów, poziom integracji, poziom systemu)?

- Jakiego rodzaju testy mają być wspierane (np. testowanie funkcjonalne, testowanie zgodności, testowanie współdziałania)?
- Która rola związana z testami ma być wspierana (np. wykonujący testy, analityk testów, architekt testów, kierownik testów)?
- Jaki produkt, linia produktów lub rodzina produktów mają być wspierane (informacja ta jest potrzebna np. do określenia zasięgu i okresu eksploatacji zaimplementowanego TAS)?
- Jakie technologie wykorzystywane w systemie podlegającym testowaniu mają być wspierane (informacja ta jest potrzebna np. do określenia wymagań stawianych TAS w zakresie kompatybilności z technologiami używanymi w systemie podlegającym testowaniu)?

Porównanie i zestawienie różnych podejść do projektowania i tworzenia architektury

Przystępując do projektowania wybranych warstw architektury automatyzacji testów, inżynier automatyzacji testów musi przeanalizować zalety i wady poszczególnych podejść, a w szczególności elementy wskazane poniżej.

Rozważania dotyczące warstwy generowania testów:

- Wybór trybu generowania testów — generowanie manualne lub automatyczne.
- Wybór sposobu generowania testów — na przykład na podstawie wymagań, na podstawie danych, na podstawie scenariuszy lub na podstawie zachowań.
- Wybór strategii generowania testów — pokrycie modelu (np. drzewa klasyfikacji) w przypadku podejść opartych na danych, pokrycie przypadków użycia/wyjątków w przypadku podejść opartych na scenariuszach, pokrycie przejść/stanów/ścieżek w przypadku podejść opartych na zachowaniach itd.
- Wybór strategii doboru testów — generowanie pełnych testów kombinatorycznych jest w praktyce niemożliwe, ponieważ mogłoby doprowadzić do „eksplozji” liczby przypadków testowych, dlatego przy generowaniu i późniejszym doborze testów należy kierować się praktycznymi kryteriami pokrycia, wagami, wynikami oceny ryzyka itd.

Rozważania dotyczące warstwy definiowania testów:

- Wybór definicji testu opartego na danych, słowach kluczowych, wzorcach lub modelu.
- Wybór notacji używanej do definiowania testów — mogą to być tabele, notacja oparta na stanach, notacja stochastyczna (ang. *stochastic notation*), notacja oparta na przepływie danych, notacja oparta na procesach biznesowych, notacja oparta na scenariuszach itd., z wykorzystaniem arkuszy kalkulacyjnych, języków testowych właściwych dla danej dziedziny, notacja testowania i nadzoru nad testami TTCN-3 (ang. *Testing and Test Control Notation*), profil UTP (ang. *UML Testing Profile*) itd.
- Wybór przewodników stylistycznych i wytycznych umożliwiających definiowanie wysokiej jakości testów.
- Wybór repozytoriów przypadków testowych (arkuszy kalkulacyjnych, baz danych, plików itd.).

Rozważania dotyczące warstwy wykonywania testów:

- Wybór narzędzia do wykonywania testów.
- Wybór podejścia interpretacyjnego (z wykorzystaniem maszyny wirtualnej) lub kompilacyjnego do implementacji procedur testowych (decyzja w tym zakresie zależy zwykle od wybranego narzędzia do wykonywania testów).
- Wybór technologii implementacji procedur testowych — może to być technologia programowania imperatywnego (np. C), funkcyjnego (np. Haskell lub Erlang), obiektowego (np. C++, C# lub Java) bądź skryptowego (np. Python lub Ruby) albo technologia związana z danym narzędziem (decyzja w tym zakresie zależy zwykle od wybranego narzędzia do wykonywania testów).
- Wybór bibliotek pomocniczych ułatwiających wykonywanie testów — na przykład bibliotek urządzeń testowych, bibliotek kodowania/dekodowania itd.

Rozważania dotyczące warstwy adaptacji testów:

- Wybór interfejsów testowych do systemu podlegającego testowaniu.
- Wybór narzędzi do stymulacji i obserwacji interfejsów testowych.
- Wybór narzędzi do monitorowania systemu podlegającego testowaniu podczas wykonywania testów.
- Wybór narzędzi do śledzenia procesu wykonywania testów (np. z uwzględnieniem parametrów czasowych wykonywania testów).

Zidentyfikowanie obszarów pozwalających wykorzystać zalety abstrakcji

Abstrakcja zapewnia niezależność technologiczną TAA, a tym samym umożliwia stosowanie tego samego zestawu testowego w różnych środowiskach testowych i w połączeniu z różnymi technologiami docelowymi. W ten sposób zwiększa przenaszalność artefaktów testowych oraz pozwala uniknąć uzależnienia TAS od produktów konkretnych dostawców. Wśród zalet abstrakcji należy również wymienić zwiększenie utrzymywalności oraz zdolności adaptacyjnej nowych lub rozwijających się technologii stosowanych w systemie podlegającym testowaniu. Ponadto, abstrakcja pomaga zwiększyć przystępność TAA (i tworzonych na jej podstawie rozwiązań dla testów automatycznych - TAS) dla osób nieposiadających wiedzy technicznej, ponieważ umożliwia dokumentowanie zestawów testowych (również w formie graficznej) i objaśnianie ich na wyższym poziomie, przez co stają się one bardziej czytelne i zrozumiałe.

Inżynier automatyzacji testów (TAE) musi omówić poziom abstrakcji, który ma zostać przyjęty w poszczególnych obszarach TAS, z interesariuszami w obszarze wytwarzania oprogramowania, zapewnienia jakości i testowania. Rozmowy te pozwolą na przykład ustalić, które interfejsy w warstwie adaptacji testów i/lub warstwie wykonywania testów należy udostępnić na zewnątrz, formalnie zdefiniować i utrzymywać w stabilnej formie przez cały okres eksploatacji TAA, a także czy ma być używana abstrakcyjna definicja testu, czy też TAA ma korzystać wyłącznie z warstwy wykonywania testów ze skryptami testowymi. Ponadto należy uzgodnić, czy proces generowania testów ma zostać uogólniony poprzez zastosowanie modeli testów i technik testowania opartego na modelu. TAE musi mieć świadomość, że wybór zaawansowanej lub prostej implementacji TAA pociąga za sobą określone kompromisy w zakresie ogólnej funkcjonalności, utrzymywalności i rozszerzalności, które trzeba mieć na uwadze decydując o poziomie abstrakcji stosowanym w ramach tej architektury.

Im wyższy jest poziom abstrakcji przyjęty w ramach TAA, tym bardziej elastycznie można rozwijać tę architekturę bądź wdrażać w niej nowe podejścia lub technologie. Wiąże się to z kosztem większych inwestycji początkowych (związanych np. z większą złożonością architektury i narzędzi do automatyzacji testów, wyższym poziomem wymaganych kwalifikacji czy dłuższym procesem uczenia się), przez co odsuwa się w czasie moment osiągnięcia progu rentowności, jednak na dłuższą metę rozwiązanie takie może okazać się opłacalne. Innym skutkiem ubocznym może być spadek wydajności TAS.

Za decyzje związane ze zwrotem z inwestycji (ang. ROI – *Return of Investment*) odpowiada kierownik automatyzacji testów (TAM), natomiast rolą inżyniera automatyzacji testów jest dostarczenie danych wejściowych do procesu analizy opłacalności w postaci analiz (ocen) technicznych i porównań poszczególnych architektur i metod automatyzacji testów (uwzględniających między innymi parametry czasowe, koszty, nakłady pracy oraz korzyści).

Zapoznanie się z technologiami stosowanymi w systemie podlegającym testowaniu oraz powiązaniem między tymi technologiami a TAS

Dostęp do interfejsów testowych systemu podlegającego testowaniu ma zasadnicze znaczenie dla automatycznego wykonywania wszelkich testów. Wyróżnia się następujące poziomy dostępu:

- dostęp na poziomie oprogramowania — na przykład połączenie systemu podlegającego testowaniu z oprogramowaniem testującym,
- dostęp na poziomie API — na przykład wywoływanie przez TAS funkcji, operacji lub metod udostępnianych przez (zdalny) interfejs programowania aplikacji,
- dostęp na poziomie protokołów — na przykład interakcja między TAS a systemem podlegającym testowaniu za pośrednictwem protokołów HTTP, TCP itd.,
- dostęp na poziomie usług — na przykład interakcja między TAS a usługami systemu podlegającego testowaniu za pośrednictwem usług internetowych, usług RESTful itp.

Inżynier automatyzacji testów (TAE) musi również wybrać odpowiedni typ interakcji architektury automatyzacji testów (TAA), który określa sposób interakcji między TAS a systemem podlegającym testowaniu w sytuacji, gdy są one rozdzielone interfejsami API, protokołami lub usługami. Można wyróżnić między innymi następujące paradygmaty:

- paradygmat sterowany zdarzeniami — w tym przypadku interakcja odbywa się poprzez wymianę zdarzeń przy użyciu magistrali zdarzeń,
- paradygmat klient-serwer — w tym przypadku interakcja odbywa się poprzez wywoływanie usług od obiektów żądających do dostawcy usług,
- paradygmat oparty na połączeniach pomiędzy pojedynczymi użytkownikami (ang. *peer-to-peer*) — w tym przypadku interakcja odbywa się poprzez wywoływanie usługi od jednego z partnerów (użytkowników).

Wybór paradygmatu zależy często od architektury systemu podlegającego testowaniu, a jednocześnie może mieć wpływ na architekturę systemu podlegającego testowaniu (SUT). W związku z tym, należy starannie przeanalizować i zaprojektować wzajemne połączenie między systemem podlegającym testowaniu a TAA, aby umożliwić w przyszłości bezpieczną interakcję między tymi środowiskami.

Zapoznanie się ze środowiskiem systemu podlegającego testowaniu (SUT)

Systemem podlegającym testowaniu może być zarówno oprogramowanie autonomiczne, jak i oprogramowanie działające wyłącznie w powiązaniu z innym oprogramowaniem (np. systemy systemów), sprzętem (np. systemy wbudowane) bądź komponentami środowiskowymi (np. systemy cyberfizyczne). W związku z tym TAS jako element konfiguracji do testów automatycznych, symuluje lub emuluje środowisko systemu podlegającego testowaniu.

Poniżej przedstawiono przykłady środowisk testowych i ich zastosowań:

- komputer z SUT (systemem podlegającym testowaniu) i TAS (rozwiązaniem dla testów automatycznych) — przydatny w przypadku testowania aplikacji,
- oddzielne komputery z systemem podlegającym testowaniu i TAS połączone w sieci — przydatne w przypadku testowania oprogramowania serwerowego,
- dodatkowe urządzenia testowe do stymulacji i obserwacji interfejsów technicznych systemu podlegającego testowaniu — przydatne na przykład w przypadku testowania oprogramowania dekodera telewizji cyfrowej,
- połączone w sieci urządzenia testowe emulujące środowisko operacyjne systemu podlegającego testowaniu — przydatne w przypadku testowania oprogramowania routera sieciowego,
- symulatory odwzorowujące działanie środowiska fizycznego systemu podlegającego testowaniu — przydatne w przypadku testowania oprogramowania wbudowanej jednostki sterującej.

Określenie czasochłonności i złożoności danej implementacji architektury testaliów

Za oszacowanie nakładów pracy związanych z realizacją TAS odpowiada kierownik automatyzacji testów (TAM), natomiast inżynier automatyzacji testów (TAE) udziela mu pomocy w postaci rzetelnych wyliczeń dotyczących czasochłonności i złożoności projektu TAA. Poniżej przedstawiono przykładowe metody szacowania i ich zastosowania:

- Szacowanie przez analogię — na przykład punkty funkcyjne, szacowanie trójpunktowe, szerokopasmowa technika delficka czy szacowanie eksperckie.
- Szacowanie z wykorzystaniem struktur podziału pracy — używanych na przykład w oprogramowaniu do zarządzania lub w szablonach projektów.
- Szacowanie parametryczne — na przykład przy użyciu modelu COCOMO (ang. *Constructive Cost Model*);
- Szacowanie na podstawie wielkości — na przykład analiza punktów funkcyjnych (ang. *Function Point Analysis*), analiza punktów historyjek (ang. *Story Point Analysis*) czy analiza przypadków użycia (ang. *Use Case Analysis*).
- Szacowanie grupowe — na przykład poker planistyczny (ang. *Planning Poker*).

Określenie łatwości użycia danej implementacji architektury testaliów

Inżynier automatyzacji testów (TAE) odpowiada zarówno za funkcjonalność TAS, jego kompatybilność z systemem podlegającym testowaniu, długofalową stabilność i możliwość rozwoju, wymagane nakłady pracy i uwarunkowania dotyczące zwrotu z inwestycji, jak i za rozwiązywanie konkretnych problemów związanych z użytecznością tego rozwiązania. W tym kontekście istotne są w szczególności następujące zagadnienia:

- projektowanie ukierunkowane na testerów,
- łatwość używania TAS,
- wsparcie przez TAS innych ról związanych z wytwarzaniem oprogramowania, zapewnieniem jakości i zarządzaniem projektem,
- sprawna organizacja TAS i nawigacja w ramach TAS oraz sprawne wyszukiwanie informacji wewnątrz i przy użyciu TAS,
- dostępność przydatnych dokumentów, podręczników i tekstów pomocy dotyczących TAS,
- praktyczne raportowanie przez TAS i na jego temat,
- projektowanie iteracyjne z uwzględnieniem informacji zwrotnych z TAS i spostrzeżeń praktycznych dotyczących TAS.

3.2.2 Podejścia do automatyzacji przypadków testowych

Przypadki testowe muszą być przekształcane w sekwencje akcji, które będą następnie wykonywane w odniesieniu do systemu podlegającego testowaniu. Każdą taką sekwencję akcji można udokumentować w procedurze testowej i/lub zaimplementować w skrypcie testowym. Oprócz akcji w przypadkach testowych dla testów automatycznych należy również zdefiniować dane testowe, które posłużą do interakcji z systemem podlegającym testowaniu, a także uwzględnić kroki weryfikacji, które pozwolą potwierdzić osiągnięcie przez ten system oczekiwanego rezultatu. Sekwencję akcji można utworzyć na kilka sposobów, które opisano poniżej:

1. Inżynier automatyzacji testów (TAE) implementuje przypadki testowe bezpośrednio w skryptach testów automatycznych. Opcja ta jest najmniej zalecana, ponieważ nie uwzględnia abstrakcji i zwiększa nakłady pracy związane z pielęgnacją.
2. TAE projektuje procedury testowe i przekształca je w skrypty testów automatycznych. Opcja ta uwzględnia abstrakcję, ale nie pozwala zautomatyzować procesu generowania skryptów testowych.
3. TAE przekształca procedury testowe w skrypty testów automatycznych przy użyciu odpowiedniego narzędzia. Opcja ta łączy abstrakcję z automatycznym generowaniem skryptów.
4. TAE korzysta z narzędzia, które generuje procedury testów automatycznych i/lub bezpośrednio przekształca modele w skrypty testowe. Opcja ta zapewnia najwyższy stopień automatyzacji.

Należy zauważyć, że wybór odpowiedniej opcji zależy w dużej mierze od kontekstu projektu. Ponadto w pewnych sytuacjach bardziej efektywne może być rozpoczęcie automatyzacji testów od zastosowania jednej z mniej zaawansowanych opcji, ponieważ opcje te są zwykle łatwiejsze do zaimplementowania. Na krótką metę może to przynieść dodatkowe korzyści, chociaż stworzone w ten sposób rozwiązanie będzie trudniejsze w pielęgnacji.

Wśród powszechnie przyjętych metod automatyzacji przypadków testowych można wymienić:

- Podejście „rejestrowanie/odtwarzanie” (ang. *capture/playback approach*), które można zastosować w przypadku opcji nr 1.
- Podejście oparte na skryptach ustrukturalizowanych, podejście sterowane danymi i podejście sterowane słowami kluczowymi, które można zastosować w przypadku opcji nr 2 lub 3.
- Testowanie oparte na modelu (w tym podejście sterowane procesem, które można zastosować w przypadku opcji nr 4).

Poniżej opisano główne koncepcje oraz zalety i wady poszczególnych podejść.

Podejście „rejestrowanie/odtwarzanie”

Główna koncepcja

Cechą charakterystyczną podejścia opartego na rejestrowaniu i odtwarzaniu jest wykorzystanie narzędzi, które rejestrują interakcje z systemem podlegającym testowaniu (SUT) podczas wykonywania sekwencji akcji zdefiniowanej w procedurze testowej. Rejestrowane są dane wejściowe, a na potrzeby późniejszego sprawdzania można również zapisywać dane wyjściowe. Podczas odtwarzania zdarzeń dostępne są różne opcje manualnego i automatycznego sprawdzania danych wyjściowych:

- kontrola manualna — tester musi obserwować dane wyjściowe systemu podlegającego testowaniu pod kątem nieprawidłowości,
- pełna kontrola — wszystkie zarejestrowane dane wyjściowe systemu muszą zostać odtworzone przez system podlegający testowaniu,
- ścisła kontrola — wszystkie zarejestrowane dane wyjściowe muszą zostać odtworzone przez system podlegający testowaniu z poziomem szczegółowości odpowiadającym poziomowi szczegółowości zapisu,
- punkty kontrolne — w określonych momentach sprawdza się tylko wybrane dane wyjściowe systemu pod kątem określonych wartości.

Zalety

Podejście oparte na rejestrowaniu i odtwarzaniu można stosować w odniesieniu do testowania na poziomie graficznego interfejsu użytkownika (GUI) i/lub API. Na początkowym etapie prac zaletą jest również łatwość konfigurowania i użytkowania.

Wady

Skrypty służące do rejestrowania i odtwarzania są trudne w pielęgnacji i rozwoju, ponieważ zarejestrowane dane dotyczące operacji wykonywanych w systemie podlegającym testowaniu są mocno uzależnione od wersji systemu, w której je zarejestrowano. Na przykład, jeśli rejestrowanie danych odbywa się na poziomie interfejsu graficznego, zmiany układu tego interfejsu mogą wpływać na skrypt testowy, nawet jeśli dotyczą jedynie położenia elementów wizualnych GUI. W związku z tym, podejścia oparte na rejestrowaniu i odtwarzaniu są podatne na zmiany.

Implementację przypadków (skryptów) testowych można rozpocząć dopiero po udostępnieniu systemu podlegającego testowaniu.

Technika skryptów liniowych

Główna koncepcja

Podobnie jak w przypadku wszystkich technik skryptowych, punktem wyjścia techniki skryptów liniowych są procedury testowe wykonywane manualnie. Procedury te nie muszą być udokumentowane — wiedzę o tym, jakie testy należy wykonać i w jaki sposób, mogą posiadać analitycy testów.

Poszczególne testy wykonuje się manualnie. Narzędzie testowe zapisuje sekwencję akcji, a w niektórych przypadkach rejestruje również dane wyjściowe wyświetlane przez system podlegający testowaniu na ekranie. W rezultacie powstaje zasadniczo jeden (zwykle duży) skrypt na każdą procedurę testową. Zapisane skrypty można edytować przy użyciu języka skryptowego używanego przez dane narzędzie, aby zwiększyć ich czytelność (np. poprzez dodanie komentarzy wyjaśniających czynności wykonywane w kluczowych momentach) bądź wprowadzić dodatkowe mechanizmy kontroli przy wykorzystaniu języka skryptowego narzędzia.

Utworzony skrypt można następnie odtworzyć za pomocą narzędzia, co powoduje powtórzenie czynności wykonanych przez testera podczas zapisywania skryptu. Technika ta może służyć do automatyzowania testów wykonywanych przy użyciu interfejsu graficznego (GUI), ale nie sprawdza się w przypadku konieczności zautomatyzowania dużej liczby testów bądź wykonywania ich w odniesieniu do wielu wersji oprogramowania. Wynika to z wysokich kosztów pielęgnacji związanych zwykle ze zmianami w systemie podlegającym testowaniu (każda zmiana w tym systemie może wymusić wprowadzenie wielu zmian w zarejestrowanych skryptach).

Zalety

Jedną z najważniejszych zalet techniki skryptów liniowych jest to, że wymagają one jedynie niewielkich (lub wręcz nie wymagają żadnych) nakładów pracy na etapie przygotowania do rozpoczęcia automatyzacji. Po opanowaniu zasad obsługi narzędzia wystarczy zarejestrować test wykonywany manualnie, a następnie go odtworzyć (choćby proces rejestrowania może wymagać dodatkowej interakcji z narzędziem testowym w celu porównania danych wyjściowych i wyników oczekiwanych, a tym samym sprawdzenia poprawności działania oprogramowania). Umiejętność programowania nie jest przy tym konieczna, chociaż zwykle okazuje się przydatna.

Wady

Technika skryptów liniowych ma wiele wad. Nakład pracy niezbędny do zautomatyzowania danej procedury testowej zależy głównie od jej rozmiaru, czyli liczby kroków lub akcji niezbędnych do jej wykonania. Oznacza to, że utworzenie tysięcznej procedury testowej wymagającej zautomatyzowania jest proporcjonalnie tak samo czasochłonne, jak utworzenie setnej procedury testowej. Innymi słowy, możliwości obniżania kosztów związanych z tworzeniem nowych testów automatycznych są bardzo ograniczone.

Jeśli występują dwa skrypty, które wykonują podobne testy, ale z użyciem innych wartości wejściowych, drugi skrypt musi zawierać tę samą sekwencję instrukcji, co pierwszy skrypt — różne są jedynie informacje zawarte w instrukcjach (zwane argumentami lub parametrami). W przypadku utworzenia kilku podobnych testów (a tym samym skryptów), każdy z nich będzie zawierał tę samą sekwencję instrukcji, która będzie wymagała aktualizacji za każdym razem, gdy w oprogramowaniu zostaną wprowadzone zmiany mające wpływ na skrypty.

Skrypty tworzy się w języku programowania, a nie w języku naturalnym, przez co mogą być niezrozumiałe dla osób niebędących programistami. Ponadto, niektóre narzędzia testowe korzystają z własnych języków (unikalnych dla danego narzędzia), których opanowanie (zwłaszcza na poziomie biegłości) wymaga czasu.

Zarejestrowane skrypty zawierają w komentarzach jedynie ogólne stwierdzenia (lub nie zawierają ich wcale). Zwłaszcza w przypadku długich skryptów zaleca się dodawanie komentarzy objaśniających czynności wykonywane w każdym kolejnym kroku testu, ponieważ ułatwia to pielęgnację. Jeśli test składa się z wielu kroków, skrypty szybko rozrastają się do bardzo dużych rozmiarów, ponieważ zawierają wiele instrukcji.

Skrypty nie mają budowy modułowej, przez co są trudne w utrzymaniu. Ponadto, proces tworzenia skryptów liniowych nie podlega powszechnie przyjętym zasadom dotyczącym ponownego wykorzystania i modułowości oprogramowania, a także jest ściśle powiązany z konkretnym narzędziem.

Technika skryptów ustrukturalizowanych

Główna koncepcja

Najważniejszą różnicą między techniką skryptów ustrukturalizowanych a techniką skryptów liniowych jest wprowadzenie biblioteki skryptów. Biblioteka ta zawiera skrypty wielokrotnego użytku, które służą do wykonywania sekwencji instrukcji wspólnych dla większej liczby testów. Dobrym przykładem są skrypty wykonujące operacje przy użyciu interfejsów systemu podlegającego testowaniu.

Zalety

Wśród zalet tego podejścia można wymienić znaczne zmniejszenie liczby zmian wymaganych na etapie utrzymania oraz obniżenie kosztów automatyzacji nowych testów (dzięki możliwości wykorzystania istniejących skryptów zamiast tworzenia od podstaw nowych skryptów).

Korzyści związane z techniką skryptów ustrukturalizowanych wynikają w dużej mierze z możliwości ponownego wykorzystania dotychczasowych skryptów. Pozwala to zautomatyzować więcej testów bez konieczności tworzenia ogromnej liczby skryptów (w odróżnieniu od podejścia opartego na skryptach liniowych), co bezpośrednio wpływa na koszty tworzenia i utrzymania testów. Dzięki możliwości ponownego wykorzystania niektórych skryptów utworzonych w celu implementacji pierwszego testu, zautomatyzowanie drugiego i kolejnych testów nie jest już tak pracochłonne.

Wady

Początkowy nakład pracy związany z tworzeniem współużytkowanych skryptów można uznać za wadę, jednak inwestycja ta powinna przynieść duże korzyści, jeśli zostanie prawidłowo przeprowadzona. Tworzenie skryptów wymaga umiejętności programowania, ponieważ samo rejestrowanie działań nie jest już wystarczające. Ponadto, konieczne jest należyte zarządzanie biblioteką skryptów: skrypty powinny być udokumentowane, a techniczni analitycy testów powinni mieć możliwość łatwego wyszukiwania wymaganych skryptów (pomaga w tym przemysłowa konwencja nazewnictwa).

Testowanie sterowane danymi

Główna koncepcja

Technika testowania sterowanego danymi jest rozwinięciem techniki skryptów ustrukturalizowanych. Najistotniejsza różnica dotyczy sposobu postępowania z danymi wejściowymi do testów. Dane te są wyodrębniane ze skryptów i umieszczane w jednym lub kilku oddzielnych plikach (zwanymi zwykle plikami danych).

Oznacza to, że główny skrypt testowy można wykorzystać do implementacji kilku, a nie tylko pojedynczego testu. Główny skrypt testowy „wielokrotnego użytku” jest zwykle nazywany skryptem sterującym. Skrypt sterujący zawiera sekwencję instrukcji niezbędnych do wykonania testów, natomiast dane wejściowe są odczytywane z pliku danych. Dzięki temu można wykorzystać jeden skrypt do przeprowadzenia wielu testów, chociaż zwykle nie wystarcza to do zautomatyzowania

szerokiej gamy testów. W związku z tym potrzebnych jest kilka skryptów sterujących, jednak ich liczba jest nadal niewielka w porównaniu z łączną liczbą zautomatyzowanych testów.

Zalety

Przedstawiona technika skryptowa pozwala znacznie obniżyć koszty związane z dodawaniem nowych testów automatycznych. Dzięki niej można zautomatyzować wiele wariantów użytecznego testu, a tym samym dokładniej przetestować dany obszar oraz zwiększyć pokrycie testowe.

W związku z tym, że testy są „opisywane” przez pliki danych, analitycy testów mogą tworzyć specyfikacje testów automatycznych poprzez wypełnienie jednego lub kilku takich plików. Metoda ta daje analitykom testów większą swobodę w zakresie tworzenia specyfikacji testów automatycznych, a przy tym nie wymaga zbyt częstego korzystania z pomocy technicznych analityków testów (których dostępność może być ograniczona).

Wady

Wadą tego podejścia jest konieczność zarządzania plikami danych oraz dbania o to, aby pliki te mogły być odczytywane przez rozwiązanie dla testów automatycznych (TAS), jednak związane z tym nakłady pracy można skutecznie ograniczyć.

Kolejną wadą jest ryzyko pominięcia negatywnych przypadków testowych. Testy negatywne stanowią połączenie procedur testowych i danych testowych, a w przypadku podejścia ukierunkowanego głównie na same dane testowe powstaje niebezpieczeństwo pominięcia „negatywnych procedur testowych”.

Testowanie oparte na słowach kluczowych

Główna koncepcja

Technika skryptów opartych na słowach kluczowych jest rozwinięciem techniki skryptów sterowanych danymi. Najważniejsze różnice polegają na tym, że po pierwsze: pliki danych są w tym przypadku nazywane plikami definicji testów lub podobnie (np. plikami słów akcji), a po drugie: występuje tylko jeden skrypt sterujący.

Plik definicji testów opisuje testy w sposób, który powinien być bardziej zrozumiały dla analityków testów (w porównaniu z równoważnym plikiem danych). Plik taki zawiera zwykle zarówno dane (analogicznie do plików danych), jak i instrukcje wysokiego poziomu (słowa kluczowe, zwane także słowami akcji).

Słowa kluczowe należy dobierać tak, aby były zrozumiałe dla analityków testów oraz adekwatne do opisywanych testów i testowanej aplikacji. Słowa te służą głównie (choć nie tylko) do odzwierciedlania interakcji biznesowych wysokiego poziomu z danym systemem (takich jak np. „złożenie zamówienia”), przy czym każde słowo kluczowe odpowiada kilku szczegółowym interakcjom z systemem podlegającym testowaniu. Przypadki testowe specyfikuje się za pomocą sekwencji słów kluczowych (wraz z odpowiednimi danymi testowymi). W przypadku kroków weryfikacji można stosować specjalne słowa kluczowe, jak również słowa kluczowe mogą określać akcje i kroki weryfikacji.

Zakres obowiązków analityków testów obejmuje tworzenie i utrzymanie plików słów kluczowych. Oznacza to, że po zaimplementowaniu skryptów pomocniczych analitycy testów mogą dodawać „automatyczne” testy, określając je po prostu w pliku słów kluczowych (podobnie jak w przypadku techniki testowania sterowanego danymi).

Zalety

W przypadku tej techniki skryptowej koszt dodawania nowych testów automatycznych ulega znacznemu obniżeniu po zakończeniu etapu tworzenia skryptu sterującego i skryptów pomocniczych (implementujących słowa kluczowe).

W związku z tym, że testy są „opisywane” przez pliki słów kluczowych, analitycy testów mogą tworzyć specyfikacje testów automatycznych poprzez opisywanie testów za pomocą słów kluczowych i związanych z nimi danych. Metoda ta daje analitykom testów większą swobodę w zakresie tworzenia specyfikacji testów automatycznych, a przy tym nie wymaga zbyt częstego korzystania z pomocy technicznych analityków testów (których dostępność może być ograniczona). Przewaga podejścia opartego na słowach kluczowych nad podejściem opartym na danych wynika właśnie

z zastosowania słów kluczowych. Każde takie słowo powinno odpowiadać sekwencji szczegółowych akcji, których wykonanie przynosi konkretny rezultat. Na przykład, w aplikacji do obsługi zakupów w Internecie mogą występować akcje takie jak „utwórz konto”, „złóż zamówienie” czy „sprawdź status zamówienia”, przy czym każda z nich może składać się z kilku szczegółowych kroków. Opisuując sobie nawzajem testy systemowe, analitycy testów będą najprawdopodobniej odwoływać się właśnie do takich akcji wysokiego poziomu, a nie szczegółowych kroków. Znajduje to odzwierciedlenie w podejściu opartym na słowach kluczowych, które umożliwia implementację akcji wysokiego poziomu i definiowanie testów na ich podstawie, bez konieczności odwoływania się do szczegółowych kroków.

Ukrycie szczegółów w słowach kluczowych (lub bibliotekach w przypadku stosowania techniki skryptów ustrukturalizowanych) ułatwia odczytywanie i zapisywanie przypadków testowych oraz ich utrzymywanie. Tym samym słowa kluczowe umożliwiają abstrakcję od złożoności interfejsów systemu podlegającego testowaniu.

Wady

Implementacja słów kluczowych pozostaje dla inżynierów automatyzacji testów trudnym zadaniem, zwłaszcza jeśli stosowane narzędzie nie obsługuje tej techniki skryptowej. W przypadku niewielkich systemów implementacja może wiązać się ze zbyt dużym nakładem pracy, a koszty mogą przewyższać korzyści.

Dobór implementowanych słów kluczowych wymaga dużej staranności. Właściwie dobrane słowa kluczowe będą wykorzystywane często w ramach wielu różnych testów, a niewłaściwie dobrane mogą zostać użyte zaledwie raz lub kilka razy.

Technika skryptów sterowanych przez proces

Główna koncepcja

Technika skryptów sterowanych przez proces jest rozwinięciem techniki skryptów opartych na słowach kluczowych. Różnica polega na tym, że rolę skryptów pełnią teraz scenariusze odpowiadające przypadkom użycia systemu podlegającego testowaniu i ich wariantom. Skrypty są następnie parametryzowane przy użyciu danych testowych lub łączone w definicje testów wyższego poziomu.

Tworzone w ten sposób definicje testów są wygodniejsze w użyciu, ponieważ pozwalają określić powiązania logiczne między akcjami (np. „sprawdź status zamówienia” poprzedzone „złóż zamówienie” w przypadku testowania funkcjonalnego lub „sprawdź status zamówienia” nie poprzedzone „złóż zamówienie” w przypadku testowania odporności).

Zalety

Zastosowanie procesowej definicji przypadków testowych, która bazuje na scenariuszach, umożliwia definiowanie procedur testowych z perspektywy przepływu pracy. Podejście oparte na procesach ma na celu implementowanie tych przepływów pracy wysokiego poziomu z wykorzystaniem bibliotek testowych, które odpowiadają szczegółowym krokom testów (patrz także podejście oparte na słowach kluczowych).

Wady

Techniczni analitycy testów mogą mieć problem ze zrozumieniem procesów zachodzących w systemie podlegającym testowaniu, a w rezultacie — z implementacją skryptów zorientowanych na procesy, zwłaszcza, jeśli wybrane narzędzie nie obsługuje logiki procesów biznesowych.

Dobór procesów (dokonywany przy użyciu słów kluczowych) wymaga dużej staranności. Do właściwie dobranych procesów będą odwoływać się inne procesy, co pozwoli utworzyć wiele testów dopasowanych do potrzeb. Niewłaściwy dobór procesów nie przyniesie korzyści pod względem dopasowania, możliwości wykrywania błędów itd.

Testowanie oparte na modelu

Główna koncepcja

Metoda testowania opartego na modelu służy do automatycznego generowania przypadków testowych (patrz także sylabus ISTQB® Testowanie oparte na modelu), nie zaś ich automatycznego wykonywania, jak w przypadku użycia techniki rejestrowania/odtworzenia, techniki skryptów liniowych, techniki skryptów ustrukturalizowanych, techniki skryptów sterowanych danymi lub techniki skryptów sterowanych przez proces. W ramach testowania opartego na modelu stosuje się (pół)formalne modele umożliwiające abstrakcję od technologii skryptowych określonych przez architekturę automatyzacji testów (TAA). Dzięki temu można skorzystać z różnych metod generowania testów w celu wyprowadzania testów odpowiadających poszczególnym omówionym powyżej strukturom skryptowym.

Zalety

Dzięki abstrakcji testowanie oparte na modelu pozwala skoncentrować się na zasadniczych aspektach testowania (takich jak logika biznesowa, dane, scenariusze, konfiguracje i inne elementy wymagające przetestowania). Ponadto podejście to umożliwia generowanie testów na potrzeby różnych systemów i technologii docelowych. Dzięki temu modele służące do generowania testów pozostaną w przyszłości użytecznym odwzorowaniem testaliów, umożliwiającym ich ponowne wykorzystanie oraz dostosowywanie do zmian technologicznych.

W przypadku zmiany wymagań wystarczy tylko dostosować do nich model testów, a następnie wygenerować automatycznie kompletny zestaw przypadków testowych (techniki projektowania przypadków testowych są wbudowane w generatory przypadków testowych).

Wady

Skuteczne stosowanie techniki testowania opartego na modelu wymaga specjalistycznej wiedzy w dziedzinie modelowania, ponieważ modelowanie bez konkretnej, szczególnej wiedzy na temat interfejsów, danych i/lub zachowań systemu podlegającego testowaniu nie jest rzeczą łatwą. Co więcej, narzędzia do modelowania i testowania opartego na modelu nie są jeszcze szeroko rozpowszechnione, chociaż z czasem stają się coraz bardziej dopracowane. Testowanie oparte na modelu wymaga również dostosowania procesów testowych, na przykład poprzez ustanowienie roli projektanta testów. Ponadto, modele służące do generowania testów są ważnymi artefaktami z punktu widzenia zapewnienia jakości systemu podlegającego testowaniu, w związku z czym muszą być również objęte procedurami kontroli jakości i utrzymania.

3.2.3. Uwarunkowania techniczne związane z systemem podlegającym testowaniu (SUT)

Przy projektowaniu architektury automatyzacji testów (TAA) należy również wziąć pod uwagę aspekty techniczne systemu podlegającego testowaniu. Poniżej omówiono niektóre z nich, przy czym podana lista nie jest kompletna i uwzględnia jedynie wybrane istotne aspekty tego zagadnienia.

Interfejsy systemu podlegającego testowaniu

System podlegający testowaniu jest wyposażony w interfejsy wewnętrzne (łącznie jego poszczególne elementy) oraz zewnętrzne (łącznie system ze środowiskiem i użytkownikami, mogące mieć postać udostępnionych komponentów). W związku z tym TAA musi umożliwiać sterowanie wszystkimi interfejsami systemu podlegającego testowaniu, na które mogą oddziaływać procedury testowe, oraz/lub obserwowanie takich interfejsów (co oznacza, że interfejsy te muszą być testowalne). Ponadto może być konieczne rejestrowanie interakcji między systemem podlegającym testowaniu a rozwiązaniem dla testów automatycznych (TAS) na różnych poziomach szczegółowości, zwykle z uwzględnieniem sygnatur czasowych (ang. *time stamps*).

Na początku projektu, na etapie definiowania architektury (a w środowiskach zwinnych również w sposób ciągły na dalszych etapach), należy określić odpowiedni przedmiot zainteresowania (np. test), aby zweryfikować dostępność interfejsów lub narzędzi testowych niezbędnych do zapewnienia testowalności systemu podlegającego testowaniu (jest to element projektowania pod kątem testowalności).

Dane związane z systemem podlegającym testowaniu

System podlegający testowaniu korzysta z danych konfiguracyjnych do sterowania procesami tworzenia instancji, konfigurowania, administrowania itd., a także z danych, które są w nim przetwarzane. Ponadto, system może wykorzystywać do realizacji swoich zadań dane zewnętrzne pochodzące z innych systemów. W związku z tym — zależnie od procedur testowych opracowanych w odniesieniu do systemu podlegającego testowaniu — należy zapewnić możliwość definiowania i konfigurowania przez TAA wszystkich powyższych typów danych oraz tworzenia ich instancji. Konkretny sposób postępowania z danymi związanymi z systemem podlegającym testowaniu ustala się w projekcie TAA. Zależnie od podejścia, dane te mogą być przetwarzane w postaci parametrów, arkuszy danych testowych, testowych baz danych, danych rzeczywistych itp.

Konfiguracje systemu podlegającego testowaniu (SUT)

System podlegający testowaniu może być wdrażany w różnych konfiguracjach, na przykład w różnych systemach operacyjnych lub urządzeniach docelowych bądź z różnymi ustawieniami językowymi. W zależności od procedur testowych może zająć potrzeba uwzględnienia powyższych różnic po stronie TAA: procedury te mogą na przykład wymagać zastosowania różnych konfiguracji testowych (w laboratorium) lub wirtualnych konfiguracji testowych (w chmurze) TAA w odniesieniu do poszczególnych konfiguracji systemu podlegającego testowaniu. Ponadto, do przetestowania określonych aspektów SUT mogą być niezbędne dodatkowe symulatory i/lub emulatory wybranych komponentów.

Normy i wymagania prawne dotyczące systemu podlegającego testowaniu (SUT)

Obok aspektów technicznych systemu podlegającego testowaniu w projekcie TAA należy również uwzględnić wymagania wynikające z obowiązujących przepisów prawa i/lub norm oraz zapewnić zgodność architektury z tymi wymaganiami. Przykładem mogą być wymagania dotyczące ochrony danych testowych lub zachowania poufności, które mają wpływ na korzystanie z oferowanych przez TAA funkcji rejestrowania i raportowania.

Narzędzia i środowiska narzędziowe używane do wytwarzania systemu podlegającego testowaniu

W związku z wytwarzaniem systemu podlegającego testowaniu mogą być używane różne narzędzia służące do opracowywania wymagań, projektowania i modelowania, tworzenia kodu, integracji oraz wdrażania. Konfiguracja TAA i jej narzędzi powinna uwzględniać powyższe środowisko narzędziowe systemu podlegającego testowaniu, ponieważ jest to konieczne do zapewnienia kompatybilności narzędzi oraz możliwości śledzenia i/lub ponownego wykorzystania artefaktów.

Interfejsy testowe w oprogramowaniu

Zdecydowanie nie zaleca się usuwania interfejsów testowych z produktu przed przekazaniem go do eksploatacji. Interfejsy te można w większości przypadków pozostawić w systemie podlegającym testowaniu bez szkody dla jego finalnej wersji, aby umożliwić pracownikom serwisu i pomocy technicznej diagnozowanie problemów oraz testowanie wersji serwisowych. Nie wolno oczywiście zapomnieć o sprawdzeniu, czy powyższe interfejsy nie będą stwarzać ryzyka w dziedzinie zabezpieczeń. W razie potrzeby programiści mogą zwykle dezaktywować interfejsy testowe, aby uniemożliwić ich użycie przez osoby spoza działu wytwarzania.

3.2.4 Uwarunkowania związane z procesami wytwarzania oprogramowania i zapewnienia jakości

Przy projektowaniu architektury automatyzacji testów (TAA) należy wziąć pod uwagę aspekty związane z procesami wytwarzania oprogramowania i zapewnienia jakości systemu podlegającego testowaniu (SUT). Poniżej omówiono niektóre z nich, przy czym podana lista nie jest kompletna i uwzględnia jedynie wybrane istotne aspekty tego zagadnienia.

Wymagania dotyczące sterowania wykonywaniem testów

Zależnie od poziomu automatyzacji wymaganego przez TAA, może okazać się konieczne zapewnienie obsługi interaktywnego wykonywania testów, wykonywania testów w trybie wsadowym bądź całkowicie automatycznego wykonywania testów.

Wymagania dotyczące raportowania

Zależnie od wymagań dotyczących raportowania (w tym typów i struktur raportów), TAA musi umożliwiać obsługę stałych, sparametryzowanych lub zdefiniowanych raportów z testów w różnych formatach i układach.

Role i prawa dostępu

Zależnie od wymagań dotyczących zabezpieczeń, może okazać się konieczne stworzenie w ramach TAA odpowiedniego systemu ról i uprawnień dostępu.

Ustalane środowisko narzędziowe

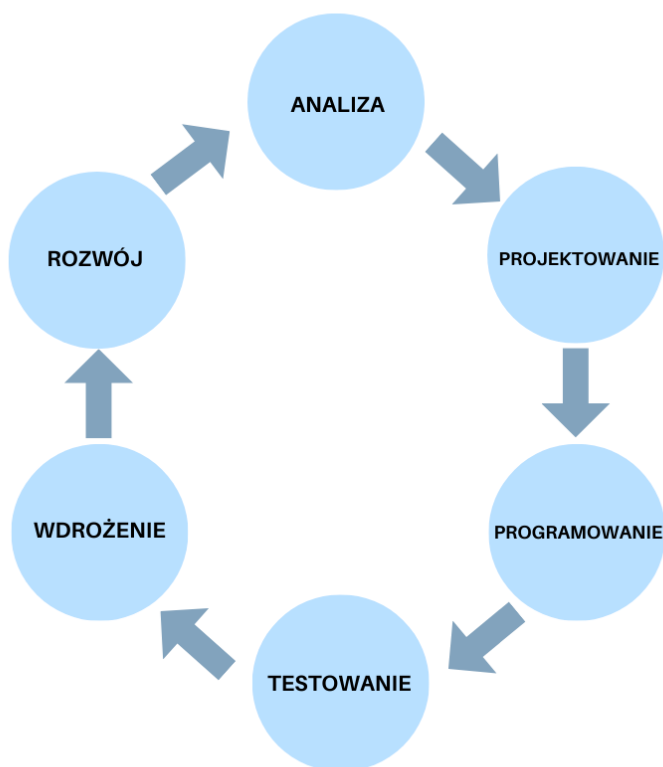
Do zarządzania projektem, zarządzania testami, obsługi repozytorium kodu i testów, śledzenia defektów, zarządzania incydentami, przeprowadzania analiz ryzyka i wykonywania innych czynności związanych z systemem podlegającym testowaniu często są używane różne narzędzia, które wspólnie tworzą ugruntowane środowisko narzędziowe. W tej sytuacji niezbędna jest możliwość bezproblemowej integracji narzędzia lub narzędzi obsługujących TAA z innymi narzędziami w powyższym środowisku. Ponadto należy zadbać o to, aby skrypty testowe były przechowywane i wersjonowane w taki sam sposób jak kod systemu podlegającego testowaniu, co pozwoli ujednolicić proces wprowadzania zmian w obu obszarach.

3.3 Tworzenie rozwiązania dla testów automatycznych (TAS)

3.3.1 Wprowadzenie do procesu tworzenia TAS

Tworzenie rozwiązania dla testów automatycznych (ang. *Test Automation Solution* - TAS) można porównać do innych projektów wytwarzania oprogramowania. W obu przypadkach prace mogą być prowadzone z wykorzystaniem tych samych procedur i procesów, łącznie z przeglądami koleżeńskimi dokonywanymi przez programistów i testerów. Cechą wyróżniającą tworzenie TAS jest konieczność zapewnienia kompatybilności i synchronizacji z systemem podlegającym testowaniu — oba te aspekty wymagają uwzględnienia zarówno w projekcie TAA (patrz podrozdział 3.2), jak i w procesie tworzenia docelowego rozwiązania. Kolejnym istotnym aspektem jest wpływ strategii testów na system podlegający testowaniu, co może na przykład wiązać się z koniecznością udostępnienia interfejsów testowych na potrzeby rozwiązania dla testów automatycznych (TAS).

W tej sekcji do omówienia procesu tworzenia TAS oraz związanych z nim aspektów dotyczących kompatybilności i synchronizacji z systemem podlegającym testowaniu (SUT) posłużono się koncepcją SDLC. Powyższe aspekty są tak samo istotne dla wszelkich innych procesów wytwarzania oprogramowania, które wybrano lub wdrożono w odniesieniu do systemu podlegającego testowaniu i/lub TAS — również i te procesy wymagają odpowiedniego dostosowania. Rysunek 2. Podstawowy cykl życia oprogramowania dotyczący TAS.



Rysunek 2. Podstawowy cykl życia oprogramowania dotyczący TAS

Na początku należy zebrać i przeanalizować zbiór wymagań dotyczących TAS (patrz Rys. 2). Wymagania te są podstawą do zaprojektowania TAS w sposób zdefiniowany przez TAA (patrz podrozdział 3.2). Następnie, przy użyciu odpowiednich metod inżynierii oprogramowania, projekt jest przekształcany w oprogramowanie. Należy przy tym zaznaczyć, że TAS może również korzystać z wydzielonego sprzętu testowego, jednak zagadnienie to nie jest przedmiotem niniejszego sylabusu. Podobnie jak każde inne oprogramowanie, TAS także wymaga przetestowania. W tym celu przeprowadza się zwykle podstawowe testy możliwości TAS, po których następuje badanie wzajemnej interakcji między TAS a systemem podlegającym testowaniu. Po wdrożeniu TAS i rozpoczęciu jego eksploatacji często konieczny jest dalszy rozwój polegający na dodawaniu kolejnych funkcji testowania, wprowadzaniu zmian w dotychczasowych testach lub aktualizowaniu rozwiązania w związku ze zmianami w systemie podlegającym testowaniu. Zgodnie z cyklem wytwarzania oprogramowania, dalszy rozwój TAS wymaga przeprowadzenia kolejnej rundy prac programistycznych.

Należy również pamiętać, że SDLC nie uwzględnia etapów tworzenia kopii zapasowych, archiwizacji i rozmontowywania TAS. Podobnie jak prace programistyczne związane z TAS, procedury te powinny być realizowane zgodnie z przyjętymi w organizacji metodami.

3.3.2. Kompatybilność TAS z systemem podlegającym testowaniu (SUT)

Kompatybilność procesów

Testowanie każdego SUT powinno być zsynchronizowane z jego wytwarzaniem, a w przypadku automatyzacji — również z tworzeniem TAS. W związku z tym, zalecane jest odpowiednie skoordynowanie procesów związanych z wytwarzaniem systemu podlegającego testowaniu, tworzeniem TAS i testowaniem. Duże korzyści może również przynieść zapewnienie kompatybilności procesów wytwarzania systemu podlegającego testowaniu i TAS pod względem struktury procesów, zarządzania procesami i obsługi narzędzi.

Koordynacja pracy zespołów

Kolejnym aspektem kompatybilności prac programistycznych związanych z TAS i systemem podlegającym testowaniu jest koordynacja pracy zespołów. Przyjęcie wspólnego podejścia do tworzenia TAS i systemu podlegającego testowaniu oraz zarządzania tymi procesami może przynieść zaangażowanym zespołom duże korzyści, takie jak możliwość wzajemnego przeglądu przyjętych wymagań, projektów i/lub artefaktów programistycznych, wspólnego omawiania problemów oraz znajdowania kompatybilnych rozwiązań. Ponadto, koordynacja pracy zespołów ułatwia wymianę informacji i wzajemną interakcję.

Kompatybilność technologii

Należy również wziąć pod uwagę kompatybilność technologii stosowanych w TAS i systemie podlegającym testowaniu. Dobrym rozwiązaniem jest uwzględnienie harmonijnego współdziałania obu systemów już na samym początku prac projektowych i implementacyjnych. Nawet jeśli okaże się to bezpośrednio niemożliwe (np. z powodu niedostępności odpowiednich rozwiązań technicznych po stronie TAS lub systemu podlegającego testowaniu), nadal może istnieć szansa na zapewnienie harmonijnego współdziałania przy użyciu adapterów, wrapperów lub innych elementów pośredniczących.

Kompatybilność narzędzi

Należy również uwzględnić kompatybilność narzędzi używanych do zarządzania, wytwarzania oprogramowania i zapewnienia jakości po stronie TAS i systemu podlegającego testowaniu. Na przykład, zastosowanie tych samych narzędzi do zarządzania wymaganiami i/lub zarządzania problemami ułatwi wymianę informacji oraz koordynację prac programistycznych dotyczących TAS i systemu podlegającego testowaniu.

3.3.3 Synchronizacja TAS z systemem podlegającym testowaniu (SUT)

Synchronizacja wymagań

Po pozyskaniu wymagań dotyczących systemu podlegającego testowaniu i rozwiązania dla testów automatycznych (TAS) należy te wymagania odpowiednio opracować. Wymagania dotyczące TAS można podzielić na dwie główne grupy: (1) wymagania odnoszące się do tworzenia TAS jako systemu opartego na oprogramowaniu, takie jak wymagania dotyczące funkcji TAS służących do projektowania testów, tworzenia specyfikacji testów, analizy rezultatów testów itd., oraz (2) wymagania odnoszące się do testowania SUT przy użyciu TAS. Powyższe tak zwane wymagania dotyczące testowania odpowiadają wymaganiom stawianym systemowi podlegającemu testowaniu i odzwierciedlają jego funkcje/właściwości SUT, które mają być testowane za pomocą TAS. Po każdej aktualizacji wymagań dotyczących systemu podlegającego testowaniu

lub TAS należy ponownie zweryfikować ich wzajemną spójność oraz sprawdzić, czy wszystkim wymaganiom dotyczącym systemu podlegającego testowaniu, które mają zostać przetestowane przy użyciu TAS, odpowiadają zdefiniowane wymagania dotyczące testowania.

Synchronizacja faz wytwarzania oprogramowania

Zapewnienie gotowości TAS w terminie umożliwiającym rozpoczęcie testowania danego SUT wymaga skoordynowania poszczególnych faz wytwarzania oprogramowania. Najbardziej efektywnym rozwiązaniem jest zsynchronizowanie faz określania wymagań, projektowania, tworzenia specyfikacji i implementacji systemu podlegającego testowaniu oraz TAS.

Synchronizacja śledzenia defektów

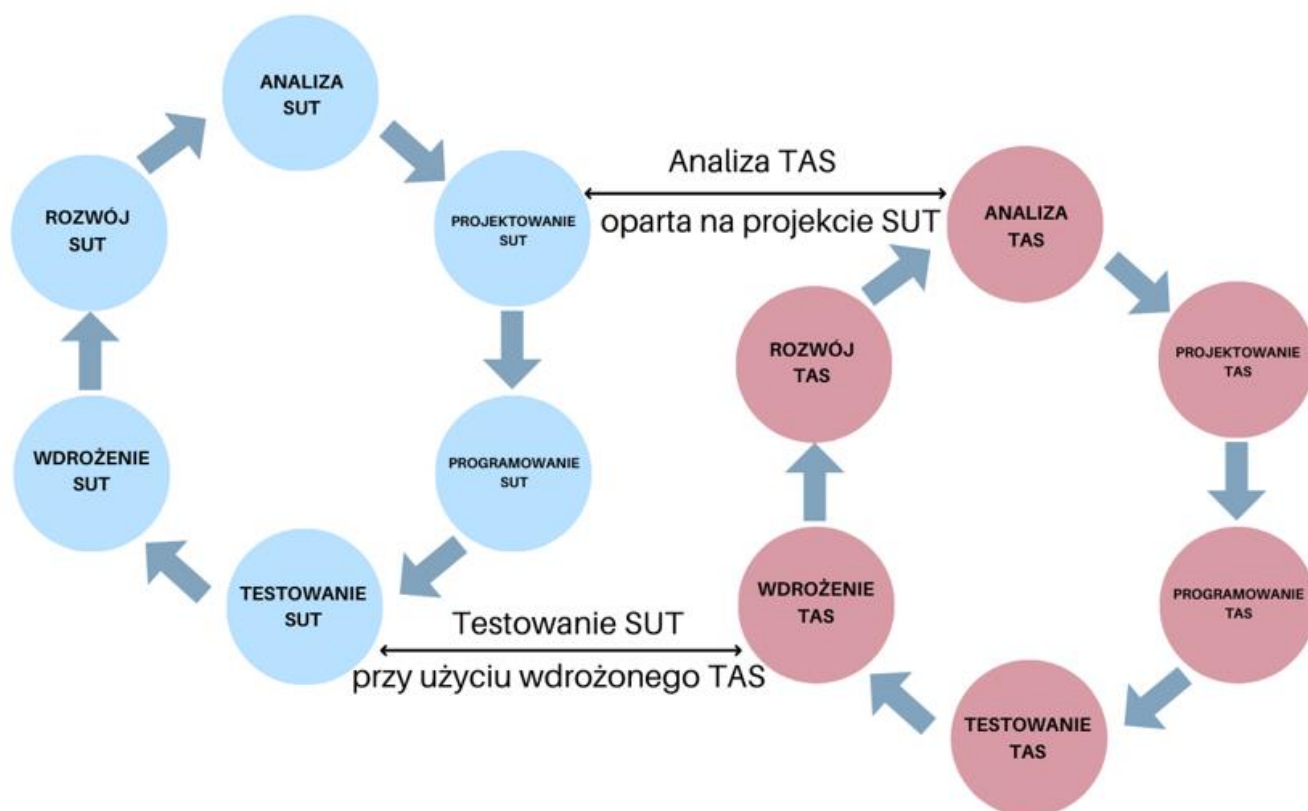
Defekty mogą dotyczyć systemu podlegającego testowaniu, TAS bądź wymagań, projektów lub specyfikacji. Z uwagi na relację między oboma projektami, działania podjęte w celu korekty defektów w jednym projekcie mogą mieć wpływ na drugi projekt, dlatego procesy śledzenia defektów i testowania potwierdzającego muszą obejmować zarówno TAS, jak i system podlegający testowaniu.

Synchronizacja rozwoju systemu podlegającego testowaniu (SUT) i TAS

SUT i TAS mogą być rozwijane w celu wprowadzania nowych funkcji lub dezaktywowania dotychczasowych, korygowania defektów bądź uwzględniania zmian w środowisku (w tym zmian wprowadzanych odpowiednio w systemie podlegającym testowaniu i TAS, ponieważ każdy z tych systemów jest elementem środowiska pracy drugiego systemu). Każda zmiana wprowadzona w systemie podlegającym testowaniu lub TAS może wpłynąć na drugi z systemów, w związku z czym zarządzanie zmianami powinno obejmować oba systemy.

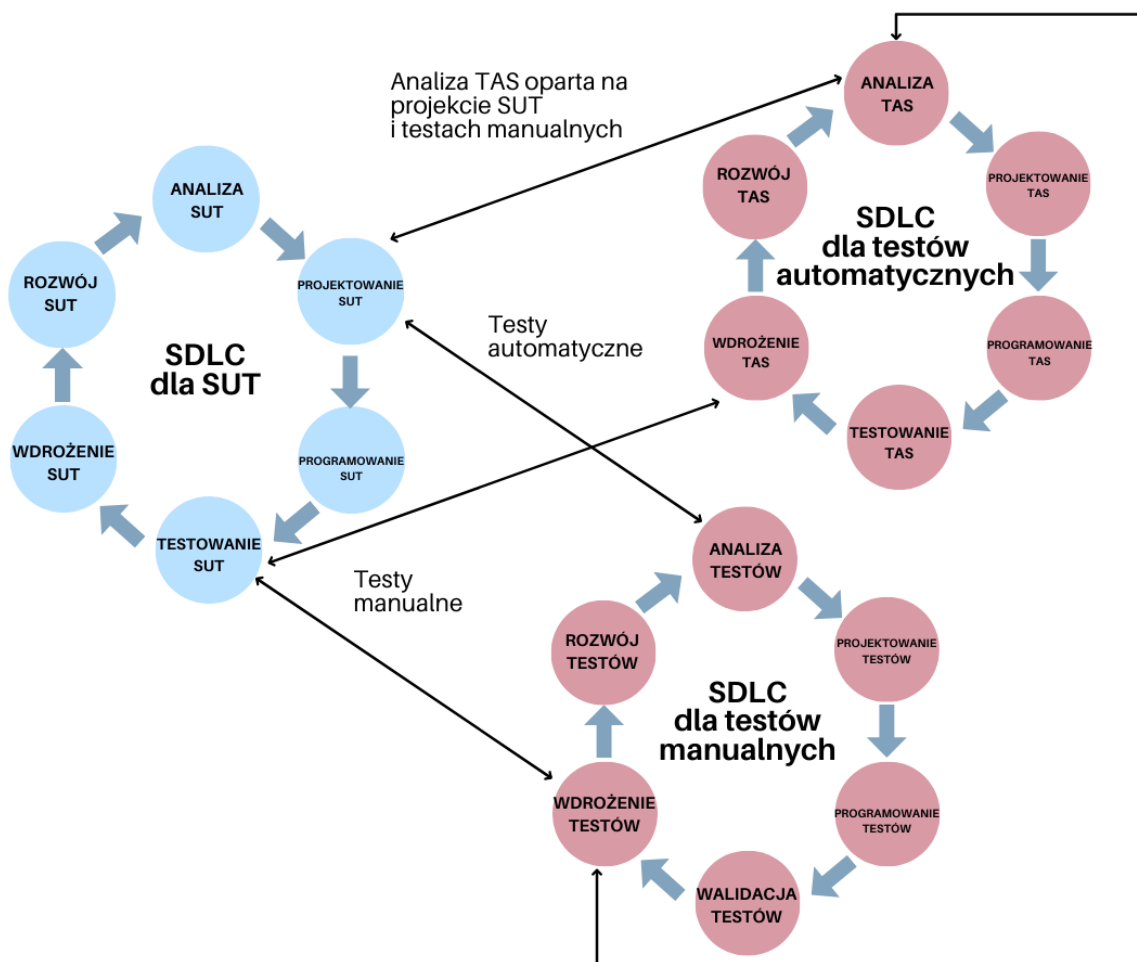
Na rysunkach 3. i 4. przedstawiono dwa podejścia do synchronizacji procesów wytwarzania oprogramowania między systemem podlegającym testowaniu a TAS.

Rysunek 3. przedstawia podejście, w którym cykle wytwarzania oprogramowania dotyczące systemu podlegającego testowaniu i TAS są zsynchronizowane głównie w dwóch fazach: (1) analiza TAS jest oparta na projekcie systemu podlegającego testowaniu, który z kolei bazuje na analizie tego systemu; (2) testy są wykonywane w systemie podlegającym testowaniu przy użyciu wdrożonego TAS.



Rysunek 3. Synchronizacja procesów wytwarzania TAS i systemu podlegającego testowaniu (SUT) — przykład nr 1

Rysunek 4. przedstawia podejście hybrydowe, które obejmuje testowanie manualne i automatyczne. Jeśli przed testami automatycznymi są stosowane testy manualne lub jeśli testy manualne i automatyczne są stosowane jednocześnie, wówczas analizę TAS należy przeprowadzić na podstawie projektu systemu podlegającego testowaniu oraz rezultatów testów manualnych. Pozwoli to zsynchronizować TAS z obydwoimi rodzajami testów. Drugi ważny punkt synchronizacji w przypadku tego podejścia jest taki sam jak poprzednio: do testowania systemu niezbędne są wdrożone testy, przy czym w przypadku testów manualnych wystarczy procedura, zgodnie z którą mają one być wykonywane.



Rysunek 4. Synchronizacja procesów wytwarzania TAS i systemu podlegającego testowaniu (SUT) — przykład nr 2

3.3.4 Projektowanie TAS pod kątem ponownego wykorzystania

W kontekście rozwiązania dla testów automatycznych (TAS) termin „ponowne wykorzystanie” dotyczy ponownego użycia artefaktów TAS (pochodzących z dowolnego poziomu jego architektury) w odniesieniu do różnych linii produktów, środowisk produktów, domen produktów i/lub rodzin produktów. Wymagania ponownego wykorzystania wynikają z potencjalnej użyteczności powyższych artefaktów z punktu widzenia innych produktów, wariantów produktów i/lub projektów.

Do artefaktów TAS nadających się do ponownego wykorzystania można zaliczyć:

- modele testów celów testów, scenariuszy testów, komponentów testów lub danych testowych (bądź ich elementy),
- przypadki testowe, dane testowe, procedury testowe lub biblioteki testowe (bądź ich elementy),
- mechanizm testowy i/lub strukturę raportów z testów,
- adaptery do komponentów i/lub interfejsów systemu podlegającego testowaniu.

Kwestie związane z ponownym wykorzystaniem ustala się już na etapie definiowania architektury automatyzacji testów (TAA), ale TAS może dodatkowo rozszerzać możliwości w tym zakresie poprzez:

- wprowadzanie w życie ustaleń przyjętych w TAA oraz ich korygowanie i aktualizowanie w zależności od bieżących potrzeb,
- dokumentowanie artefaktów TAS w sposób pozwalający łatwo je zrozumieć i wykorzystać w nowych kontekstach,
- zapewnienie poprawności wszystkich artefaktów TAS tak, aby ich wysoka jakość sprzyjała wykorzystaniu w nowych kontekstach.

Należy przy tym zaznaczyć, że o ile kwestię projektowania pod kątem ponownego wykorzystania reguluje głównie TAA, o tyle o pielęgnację i doskonalenie mechanizmów umożliwiających ponowne wykorzystanie należy dbać przez cały cykl życia TAS. Zapewnienie ponownego wykorzystania, mierzenie i wykazywanie wynikających z tego korzyści oraz przekonywanie innych do ponownego korzystania z istniejących rozwiązań do automatyzacji testów wymaga ciągłej uwagi i nieustannych wysiłków.

3.3.5. Obsługa różnych systemów docelowych

Obsługa różnych systemów docelowych oznacza możliwość wykorzystania TAS do testowania różnych konfiguracji wytwarzanego oprogramowania. Określenie „różne konfiguracje” może odnosić się do:

- liczby komponentów systemu podlegającego testowaniu i wzajemnych połączeń między nimi,
- środowisk (programowych i sprzętowych), w których działają komponenty systemu podlegającego testowaniu,
- technologii, języków programowania lub systemów operacyjnych używanych do implementacji komponentów systemu podlegającego testowaniu,
- bibliotek i pakietów wykorzystywanych przez komponenty systemu podlegającego testowaniu,
- narzędzi używanych do implementacji komponentów systemu podlegającego testowaniu.

Pierwsze cztery aspekty są istotne z punktu widzenia TAS na każdym poziomie testów, natomiast ostatni dotyczy głównie testów na poziomie modułów i integracji.

O ile możliwość wykorzystania TAS do testowania różnych konfiguracji oprogramowania określa się na etapie definiowania TAA, o tyle mechanizmy umożliwiające zapanowanie nad rozbieżnościami technicznymi oraz zarządzanie funkcjami i komponentami niezbędnymi do obsługi różnych konfiguracji oprogramowania muszą zostać zaimplementowane na poziomie TAS.

Problem zarządzania zmiennością TAS w relacji do zmienności oprogramowania można rozwiązać na różne sposoby:

- Zarządzanie wersjami/konfiguracją w odniesieniu do TAS i systemu podlegającego testowaniu pozwala zapewnić wzajemne dopasowanie wersji i konfiguracji obu systemów.
- Parametryzacja TAS pozwala na dostosowanie TAS do konfiguracji systemu podlegającego testowaniu.

Należy przy tym zaznaczyć, że o ile kwestię projektowania pod kątem zmienności TAS reguluje głównie TAA, o tyle o pielęgnację i doskonalenie mechanizmów związanych ze zmiennością należy dbać przez cały cykl życia TAS. Korygowanie, dodawanie, a nawet usuwanie opcji związanych ze zmiennością wymaga ciągłej uwagi i nieustannych wysiłków.

4. Ryzyka i sytuacje awaryjne związane z wdrożeniem — 150 min.

Słowa kluczowe

ryzyko, łagodzenie ryzyka, ocena ryzyka, ryzyko produktowe

Cele nauczania — ryzyka i sytuacje awaryjne związane z wdrożeniem

4.1 Wybór podejścia do automatyzacji testów oraz planowanie wdrożenia/wydania na rynek

ALTA-E-4.1.1 (K3) Kandydat potrafi stosować wytyczne dotyczące sprawnego wykonywania czynności związanych z pilotażem i wdrożeniem narzędzia testowego.

4.2 Strategie oceny i łagodzenia ryzyka

ALTA-E-4.2.1 (K4) Kandydat potrafi przeanalizować ryzyka związane z wdrożeniem, zidentyfikować problemy techniczne zagrażające pomyślnej realizacji projektu automatyzacji testów oraz zaplanować strategie łagodzenia ryzyka.

4.3 Utrzymanie mechanizmów automatyzacji testów

ALTA-E-4.3.1 (K2) Kandydat zna czynniki mające wpływ na utrzymywalność TAS i sprzyjające jej zwiększaniu.

4.1 Wybór podejścia do automatyzacji testów oraz planowanie wdrożenia/wydania na rynek

Dwie najważniejsze czynności wykonywane w ramach implementacji i wydania na rynek rozwiązania dla testów automatycznych (TAS) to pilotaż i wdrożenie. Kroki, z których składają się te czynności, różnią się w zależności od typu TAS oraz specyfiki danej sytuacji.

W przypadku pilotażu należy uwzględnić co najmniej następujące kroki:

- zidentyfikowanie odpowiedniego projektu,
- zaplanowanie pilotażu,
- przeprowadzenie pilotażu,
- dokonanie oceny wyników pilotażu.

W przypadku wdrożenia należy uwzględnić co najmniej następujące kroki:

- zidentyfikowanie początkowego projektu docelowego (lub projektów docelowych),
- wdrożenie TAS w ramach wybranego projektu (lub projektów),
- monitorowanie i dokonywanie oceny TAS w ramach projektów po upływie wcześniej zdefiniowanego okresu,
- wydanie na całą organizację lub inne projekty.

4.1.1 Projekt pilotażowy

Implementację narzędzia zaczyna się zwykle od przeprowadzenia projektu pilotażowego, którego celem jest upewnienie się, że zastosowanie TAS przyniesie zaplanowane korzyści. Projekt pilotażowy ma na celu między innymi:

- Uzyskanie dokładniejszych informacji na temat TAS.
- Sprawdzenie dopasowania TAS do istniejących procesów, procedur i narzędzi oraz zidentyfikowanie ewentualnie wymaganych zmian (zwykle preferowane jest dopasowanie TAS do istniejących procesów/procedur, a jeśli zachodzi potrzeba dostosowania procesów/procedur do TAS, zmiany powinny mieć charakter udoskonaleń).
- Zaprojektowanie interfejsu automatyzacji zgodnie z potrzebami testerów.
- Podjęcie decyzji co do standardowych sposobów używania, przechowywania i utrzymania TAS i zasobów testowych oraz zarządzania nimi, w tym integracji z procesami zarządzania konfiguracją i zarządzania zmianami (np. ustalenie konwencji nazewnictwa plików i testów, utworzenie bibliotek oraz określenie modułowości zestawów testów).
- Zidentyfikowanie metryk i metod pomiaru używanych do monitorowania mechanizmów automatyzacji testów podczas eksploatacji (w tym do określania ich użyteczności, utrzymywalności i rozszerzalności).
- Oszacowanie, czy można osiągnąć spodziewane korzyści uzasadnionym kosztem (jest to okazja do skorygowania oczekiwań po rozpoczęciu korzystania z TAS).
- Określenie niezbędnych umiejętności oraz ustalenie, które z nich są dostępne lub wymagają uzupełnienia.

Zidentyfikowanie odpowiedniego projektu

Wyboru projektu pilotażowego należy dokonać z zachowaniem dużej staranności, zgodnie z następującymi wytycznymi:

- Nie należy wybierać projektu mającego znaczenie krytyczne. Wdrożenie TAS może powodować opóźnienia, dlatego należy unikać sytuacji, w których opóźnienia takie mogłyby wpłynąć w istotny sposób na newralgiczne projekty (w początkowej fazie wdrożenia TAS jest czasochłonne, o czym musi pamiętać zespół projektowy).
- Nie należy wybierać prostego i mało istotnego projektu. Projekty takie nie nadają się do wykorzystania w ramach pilotażu, ponieważ powodzenie ich wdrożenia wcale nie musi przekładać się na pomyślną realizację bardziej zaawansowanego projektu, a wartość uzyskanych w ten sposób informacji jest niewielka.
- Należy zaangażować w proces wyboru niezbędnych interesariuszy (w tym kierownictwo).

- System podlegający testowaniu, którego ma dotyczyć projekt pilotażowy, powinien być dobrym punktem odniesienia dla innych projektów realizowanych przez organizację (np. powinien zawierać reprezentatywne komponenty interfejsu graficznego wymagające zautomatyzowania).

Zaplanowanie pilotażu

Pilotaż należy potraktować jako zwykły projekt wytwarzania oprogramowania, co oznacza konieczność opracowania planu, zarezerwowania środków w budżecie i zasobów, raportowania o postępach, zdefiniowania kamieni milowych itd. Szczególną uwagę należy zwrócić na to, aby osoby pracujące nad wdrożeniem TAS (tj. specjaliści) mogły poświęcić na to wystarczająco dużo czasu, nawet jeśli muszą również wykonywać pracę w ramach innych projektów. Dlatego ważne jest zaangażowanie kierownictwa, zwłaszcza gdy chodzi o pracowników dzielących swój czas między różne zadania, którzy prawdopodobnie nie będą mogli zajmować się wdrożeniem w pełnym wymiarze godzin.

Jeśli rozwiązanie do automatyzacji testów zostało opracowane we własnym zakresie (a nie dostarczone przez zewnętrznego dostawcę), w prace wdrożeniowe należy zaangażować programistów, którzy brali udział w tworzeniu TAS.

Przeprowadzenie pilotażu

Podczas pilotażu poprzedzającego wdrożenie należy zwrócić uwagę na następujące kwestie:

- Czy TAS udostępnia oczekiwane (i deklarowane przez dostawcę) funkcjonalności? Jeśli nie, należy jak najszybciej rozwiązać ten problem. W przypadku opracowywania TAS we własnym zakresie odpowiedni programiści powinni zadbać o udostępnienie brakujących funkcjonalności.
- Czy TAS i istniejący proces wzajemnie wspomagają swoje działanie? Jeśli nie, konieczne jest dopasowanie ich do siebie.

Dokonanie oceny wyników pilotażu

W procesie oceny powinni wziąć udział wszyscy interesariusze.

4.1.2 Wdrożenie

Po dokonaniu oceny wyników pilotażu i uznaniu, że zakończył się on sukcesem, można przystąpić do wdrożenia TAS w całym dziale lub całej organizacji. Wydanie należy przeprowadzić metodą przyrostową, dbając o należyte zarządzanie tym procesem. Poniżej wymieniono niektóre czynniki decydujące o sukcesie wdrożenia:

- Wydanie metodą przyrostową. Wydanie w pozostałych częściach organizacji należy przeprowadzić etapami, w sposób przyrostowy. Dzięki temu nowi użytkownicy będą uzyskiwać dostęp do rozwiązania „falami”, a nie wszyscy naraz. Stopniowe zwiększanie liczby osób korzystających z TAS pozwala zidentyfikować i wyeliminować ewentualne wąskie gardła, zanim staną się realnym problemem, a także dodawać w miarę potrzeb kolejne licencje.
- Dostosowywanie i udoskonalanie procesów pod kątem współpracy z TAS. Gdy z rozwiązania dla testów automatycznych (TAS) korzystają różni użytkownicy, rozwiązanie to wchodzi w interakcję z różnymi procesami. W rezultacie może być konieczne dostosowanie takich procesów do współpracy z rozwiązaniem lub wprowadzenie w rozwiązaniu (niewielkich) zmian dostosowujących je do istniejących procesów.
- Zapewnienie nowym użytkownikom szkoleń oraz coachingu/mentoringu. Nowi użytkownicy potrzebują szkoleń i coachingu w zakresie korzystania z nowego TAS, w związku z czym należy zadbać o dostępność odpowiednich zasobów. Szkolenia/warsztaty dla użytkowników powinny zostać przeprowadzone jeszcze przed rozpoczęciem korzystania z TAS.
- Określenie wytycznych dotyczących użytkownika. Opracowanie wytycznych, list kontrolnych i odpowiedzi na często zadawane pytania związane z korzystaniem z TAS pozwala uniknąć nadmiernej liczby zgłoszeń do działu wsparcia technicznego.
- Wprowadzenie mechanizmu zbierania informacji na temat faktycznego użytkownika. Należy zapewnić możliwość automatycznego zbierania informacji na temat faktycznego korzystania z TAS. W miarę możliwości, zbierane powinny być informacje dotyczące zarówno sposobu korzystania z rozwiązania, jak i używanych elementów (tj. konkretnych funkcjonalności). Dzięki temu łatwiej będzie monitorować eksploatację TAS.
- Monitorowanie wykorzystania TAS oraz odniesionych korzyści i poniesionych kosztów. Monitorowanie użycia TAS w określonym przedziale czasu pozwala stwierdzić, czy rozwiązanie to jest faktycznie wykorzystywane. Uzyskane informacje mogą również posłużyć do ponownego obliczenia uzasadnienia biznesowego (np. w kategoriach zaoszczędzonego czasu bądź liczby problemów, którym udało się zapobiec).

- Zapewnienie zespołom testerów i programistów wsparcia technicznego w zakresie danego TAS.
- Zbieranie doświadczeń od wszystkich zespołów. Należy organizować spotkania ewaluacyjne i retrospektywne z udziałem poszczególnych zespołów korzystających z TAS. Dzięki temu można będzie wyciągnąć wnioski z dotychczasowych doświadczeń, a zespoły będą miały poczucie, że ich opinie są mile widziane i niezbędne do usprawniania pracy z TAS.
- Identyfikowanie i implementowanie udoskonaleń. Na podstawie informacji zwrotnych od zespołów oraz danych z monitorowania TAS należy identyfikować i wdrażać odpowiednie udoskonaleń, informując o tym w przejrzysty sposób odpowiednich interesariuszy.

4.1.3 Wdrożenie TAS w kontekście cyklu życia oprogramowania

Przebieg wdrożenia TAS zależy w ogromnej mierze od fazy wytwarzania oprogramowania, które ma być testowane przy jego użyciu.

Wdrożenie nowego TAS (lub nowej wersji dotychczasowego TAS) ma zwykle miejsce albo na początku projektu, albo w momencie osiągnięcia określonego kamienia milowego — takiego jak zamrożenie kodu czy koniec sprintu. Wynika to z faktu, że działania związane z wdrożeniem (w tym wszystkie niezbędne testy i modyfikacje) wymagają nakładu czasu i pracy. Ponadto podejście takie pozwala skutecznie ograniczyć ryzyko niewłaściwego działania TAS i powstania przez to zakłóceń w funkcjonowaniu procesu automatyzacji testów. Jeśli jednak konieczne jest rozwiązanie problemów krytycznych związanych z TAS lub dokonanie wymiany komponentu środowiska, w którym działa to rozwiązanie, wdrożenie przeprowadza się niezależnie od fazy rozwoju systemu podlegającego testowaniu.

4.2 Strategie oceny i łagodzenia ryzyka

Problemy techniczne mogą prowadzić do powstania ryzyka produktowego lub projektowego. Wśród typowych problemów technicznych można wymienić:

- Zbyt wysoki poziom abstrakcji, który może utrudnić zrozumienie faktycznie zachodzących procesów (np. w przypadku korzystania ze słów kluczowych).
- Zbyt obszerne, złożone lub trudne w obsłudze tabele danych (w przypadku techniki skryptów sterowanych danymi).
- Uzależnienie rozwiązań dla testów automatycznych (TAS) od konkretnych bibliotek systemowych lub innych komponentów, które mogą nie być dostępne we wszystkich środowiskach docelowych systemu podlegającego testowaniu.

Do typowych ryzyk projektowych związanych z wdrożeniem należą:

- problemy kadrowe, czyli trudności z pozyskaniem właściwych osób do pielęgnacji kodu,
- nieprawidłowe działanie TAS spowodowane dostarczeniem nowych produktów związanych z systemem podlegającym testowaniu,
- opóźnienia we wdrażaniu automatyzacji,
- opóźnienia w aktualizowaniu TAS stosownie do zmian wprowadzanych w systemie podlegającym testowaniu,
- brak możliwości rejestrowania przez TAS obiektów (niestandardowych), które powinny być z założenia objęte śledzeniem.

Wśród potencjalnych przyczyn niepowodzenia projektu związanego z TAS można wymienić:

- migrację do innego środowiska,
- wdrożenie w nowym środowisku docelowym,
- dostarczenie przez programistów nowego elementu.

W powyższych przypadkach można zastosować kilka strategii łagodzenia ryzyka, które omówiono poniżej.

Rozwiązanie dla testów automatycznych (TAS) ma własny cykl życia — niezależnie od tego, czy jest tworzone przez organizację we własnym zakresie, czy też zostało nabyte od dostawcy. W związku z tym należy pamiętać, że rozwiązanie to (podobnie jak każde inne oprogramowanie) musi być objęte kontrolą wersji, a jego funkcje muszą być dokumentowane. W przeciwnym razie bardzo trudno będzie wdrażać poszczególne elementy TAS i zapewnić ich współdziałanie (bądź działanie w określonych środowiskach).

Ponadto, niezbędna jest udokumentowana, przejrzysta i łatwa do przestrzegania procedura wdrożeniowa. Procedura ta jest przypisana do konkretnej wersji rozwiązania, w związku z czym musi być również objęta kontrolą wersji.

Wyróżnia się dwa warianty wdrożenia TAS:

1. Wdrożenie początkowe.
2. Wdrożenie związane z pielęgnacją (przeprowadzane w sytuacji, w której TAS już istnieje i wymaga wykonania czynności związanych z pielęgnacją).

Przed rozpoczęciem pierwszego wdrożenia TAS należy upewnić się, że rozwiązanie to działa prawidłowo we własnym środowisku, jest odporne na losowe zmiany oraz umożliwia aktualizowanie przypadków testowych i zarządzanie nimi. Zarówno TAS, jak i jego infrastruktura muszą być utrzymywane.

W przypadku pierwszego wdrożenia należy wykonać następujące podstawowe kroki:

- zdefiniować infrastrukturę, w której ma działać TAS,
- stworzyć infrastrukturę na potrzeby TAS,
- opracować procedurę utrzymania TAS i jego infrastruktury,
- opracować procedurę utrzymania zestawu testów, który będzie wykonywany za pomocą TAS.

Poniżej opisano ryzyka związane z pierwszym wdrożeniem:

- Łączny czas wykonywania zestawu testowego może być dłuższy niż planowany czas wykonywania cyklu testowego. W takim przypadku należy koniecznie zarezerwować odpowiednią ilość czasu pozwalającą wykonać cały zestaw testowy przed rozpoczęciem kolejnego zaplanowanego cyklu testowego.
- W środowisku testowym mogą wystąpić problemy instalacyjne i konfiguracyjne (związane np. z konfigurowaniem i początkowym ładowaniem bazy danych bądź uruchamianiem/zatrzymywaniem usług). Zasadniczo TAS wymaga mechanizmu umożliwiającego sprawne konfigurowanie elementów niezbędnych do działania zautomatyzowanych przypadków testowych w środowisku testowym.

W przypadku wdrożeń związanych z pielęgnacją należy uwzględnić dodatkowe czynniki. TAS musi stale się rozwijać, co wiąże się z koniecznością wdrażania aktualizacji w środowisku eksploatacyjnym. Przed przekazaniem zaktualizowanej wersji TAS do eksploatacji należy to rozwiązanie przetestować — tak samo, jak każde inne oprogramowanie. W związku z tym, konieczne jest sprawdzenie nowej funkcjonalności w celu potwierdzenia, że w zaktualizowanym TAS można uruchomić odpowiedni zestaw testowy i wysłać raporty, oraz że nie występują problemy związane z wydajnością ani inne problemy związane z regresją funkcjonalną. W niektórych przypadkach może być konieczne zmodyfikowanie całego zestawu testowego w celu dopasowania go do nowej wersji TAS.

W ramach wdrożenia związanego z pielęgnacją należy wykonać następujące kroki:

- dokonać oceny zmian wprowadzonych w nowej wersji TAS w porównaniu ze starą wersją,
- przetestować TAS pod kątem nowej funkcjonalności i regresji,
- sprawdzić, czy zestaw testowy wymaga dostosowania do nowej wersji TAS.

Z aktualizacją wiąże się również następujące ryzyka i odpowiadające im działania służące łagodzeniu ryzyk:

- Konieczność zmodyfikowania zestawu testowego w celu uruchomienia go w zaktualizowanym TAS. Należy wprowadzić w zestawie testowym niezbędne zmiany, a następnie przetestować je przed wdrożeniem w TAS.
- Konieczność zmiany zaślepek, sterowników i interfejsów używanych do testowania w celu dopasowania ich do zaktualizowanego TAS. Należy wprowadzić niezbędne zmiany w jarzmie testowym, a następnie przetestować je przed wdrożeniem w TAS.
- Konieczność zmiany infrastruktury na potrzeby zaktualizowanego TAS. Należy dokonać oceny komponentów infrastruktury, które wymagają zmiany, a następnie wprowadzić niezbędne zmiany i przetestować je w połączeniu ze zaktualizowanym TAS.
- Dodatkowe defekty lub problemy związane z wydajnością w zaktualizowanym TAS. Należy przeprowadzić analizę ryzyka i korzyści. Jeśli okaże się, że wykryte problemy uniemożliwiają aktualizację TAS, najlepszym wyjściem może być rezygnacja z aktualizacji lub poczekanie na następną wersję TAS. Jeśli jednak powyższe problemy mają znikome znaczenie w porównaniu z potencjalnymi korzyściami, można dokonać aktualizacji TAS. W takim przypadku należy koniecznie przygotować opis wydania zawierający informację o znanych problemach, a także powiadomić

inżynierów automatyzacji testów i innych interesariuszy oraz postarać się uzyskać informację o przewidywanym terminie rozwiązania problemów.

4.3 Pielęgnacja rozwiązania dla automatyzacji testów

Tworzenie rozwiązań dla testów automatycznych (TAS) nie jest rzeczą prostą. Rozwiązania takie muszą mieć budowę modułową, a do tego muszą być skalowalne, zrozumiałe, niezawodne i testowalne. Sprawę dodatkowo komplikuje fakt, że rozwiązania tego typu — podobnie jak inne systemy oprogramowania — muszą stale się rozwijać. Z tego powodu ważnym aspektem projektowania architektury TAS jest pielęgnacja — niezależnie od tego, czy aktualizacje wynikają ze zmian wewnętrznych czy zmian w środowisku pracy. Aktualizacje TAS polegające na dostosowywaniu go do nowych typów testowanych systemów, wprowadzaniu obsługi nowych środowisk oprogramowania bądź zapewnianiu zgodności z nowymi przepisami prawa pomagają zagwarantować niezawodne i bezpieczne działanie, a także zoptymalizować wydajność i czas eksploatacji.

4.3.1 Rodzaje pielęgnacji

Czynności związane z pielęgnacją wykonuje się w odniesieniu do istniejącego, działającego TAS w związku z modyfikacją, migracją lub wycofaniem systemu podlegającego testowaniu. Pielęgnację można podzielić na następujące kategorie:

- Pielęgnacja prewencyjna. Zmiany wprowadzane w TAS mają na celu obsługę dodatkowych typów testów, wykonywanie testów przy użyciu wielu interfejsów, testowanie wielu wersji systemu podlegającego testowaniu bądź zautomatyzowanie testów nowego systemu podlegającego testowaniu.
- Pielęgnacja naprawcza. Zmiany są wprowadzane w celu usunięcia awarii występujących w TAS. Najlepszą metodą utrzymania aktualnie eksploatowanego TAS (pozwalającą zmniejszyć ryzyko związane z jego użytkowaniem) jest wykonywanie regularnych testów pielęgnacyjnych.
- Pielęgnacja doskonaląca. Proces ten obejmuje optymalizowanie TAS i rozwiązywanie problemów w obszarach niefunkcyjnych. Wykonywane działania mogą dotyczyć wydajności, użyteczności, odporności lub niezawodności rozwiązania.
- Pielęgnacja adaptacyjna. W miarę wprowadzania na rynek nowych systemów oprogramowania (tj. systemów operacyjnych, systemów zarządzania bazami danych, przeglądarek internetowych itd.) może być konieczne zmodyfikowanie TAS pod kątem ich obsługi. Ponadto, może zająć potrzeba zapewnienia zgodności TAS z nowymi przepisami prawa bądź wymaganiami branżowymi, co wiąże się z wprowadzeniem w tym rozwiązaniu odpowiednich zmian. Uwaga: w celu zapewnienia zgodności z przepisami zwykle konieczne jest wykonywanie obowiązkowych czynności związanych z pielęgnacją, które podlegają określonym zasadom i wymaganiom, a niekiedy są również objęte obowiązkiem przeprowadzenia audytu. Ponadto, w związku z aktualizacją i tworzeniem nowych wersji narzędzi integrujących konieczne jest utrzymanie i zapewnienie nieprzerwanego działania narzędzia w miejscach integracji z innymi narzędziami oraz systemami.

4.3.2. Zakres i podejście

Pielęgnacja jest procesem, który może wpływać na wszystkie warstwy i komponenty TAS. Jego zakres zależy od:

- wielkości i złożoności TAS,
- wielkości wprowadzanych zmian,
- ryzyka związanego ze zmianami.

Z uwagi na to, że utrzymanie dotyczy aktualnie eksploatowanego TAS, konieczne jest przeprowadzenie analizy wpływu mającej na celu ustalenie, w jaki sposób wprowadzane zmiany mogą wpłynąć na funkcjonowanie systemu. Zależnie od szacowanego wpływu, może zająć potrzeba wprowadzania zmian metodą przyrostową i przeprowadzania testów po zakończeniu każdego kroku w celu zagwarantowania nieprzerwanej pracy TAS. Uwaga: pielęgnacja TAS może być trudna w przypadku korzystania z nieaktualnych specyfikacji i dokumentacji.

Efektywne wykorzystanie czasu jest głównym czynnikiem sukcesu automatyzacji testów, dlatego niezwykle ważne jest stosowanie dobrych praktyk w dziedzinie utrzymania TAS. Poniżej wymieniono kilka ważnych zasad:

- należy jednoznacznie określić i udokumentować procedury dotyczące wdrażania i użytkowania TAS,
- należy udokumentować zależności od produktów innych firm, w tym wynikające z nich utrudnienia i zidentyfikowane wcześniej problemy,
- TAS musi mieć budowę modułową umożliwiającą łatwą wymianę poszczególnych elementów,
- TAS musi działać w środowisku, które można zastąpić innym środowiskiem lub które składa się z wymiennych komponentów,
- TAS musi umożliwiać oddzielenie skryptów testowych od struktury testów automatycznych (TAF),
- TAS musi działać niezależnie od środowiska programistycznego, tak aby zmiany w TAS nie wpływały niekorzystnie na środowisko testowe,
- TAS wraz z całym środowiskiem, zestawem testów i testaliami musi być objęte procesem zarządzania konfiguracją.

Należy również wziąć pod uwagę uwarunkowania związane z utrzymaniem komponentów innych firm oraz innych bibliotek, które wymieniono poniżej:

- TAS bardzo często wykorzystuje do wykonywania testów komponenty innych firm. Ponadto, działanie TAS może być uzależnione od bibliotek innych firm (np. bibliotek automatyzacji interfejsu użytkownika). W związku z tym, wszystkie komponenty innych firm wchodzące w skład rozwiązania muszą być udokumentowane i objęte procesem zarządzania konfiguracją.
- Niezbędne jest opracowanie planu na wypadek konieczności zmodyfikowania lub skorygowania komponentów innych firm. W związku z tym, osoba odpowiedzialna za utrzymanie TAS musi wiedzieć, z kim należy się w takiej sytuacji skontaktować lub gdzie należy zgłosić problem.
- Niezbędna jest dokumentacja dotycząca licencji, na podstawie których są wykorzystywane komponenty innych firm, zawierająca informacje o tym, czy komponenty te mogą być modyfikowane, a jeśli tak, to w jakim stopniu i przez kogo.
- Należy na bieżąco pozyskiwać informacje o aktualizacjach i nowych wersjach wszystkich komponentów innych firm. Ciągła aktualizacja komponentów i bibliotek dostarczanych przez inne firmy jest działaniem prewencyjnym przynoszącym długofalowe korzyści.

W zakresie standardów nazewnictwa i innych konwencji należy wziąć pod uwagę następujące uwarunkowania:

- Cel stosowania standardów nazewnictwa i innych konwencji jest bardzo prosty: zestaw testów i TAS muszą być czytelne, zrozumiałe i łatwe w utrzymaniu, a także muszą umożliwiać łatwe wprowadzanie zmian. Dzięki temu można zaoszczędzić czas w procesie pielęgnacji oraz zminimalizować ryzyko regresji lub wprowadzenia niewłaściwych poprawek.
- Stosowanie standardowych konwencji nazewnictwa ułatwia włączanie nowych osób do projektu automatyzacji testów.
- Standardy nazewnictwa mogą dotyczyć zmiennych i plików, scenariuszy testowych, słów kluczowych oraz parametrów słów kluczowych. Oprócz tego można stosować inne konwencje dotyczące na przykład wymagań początkowych i działań podejmowanych po wykonaniu testów, zawartości danych testowych, środowiska testowego, statusu wykonywania testów oraz dzienników i raportów dotyczących wykonania testów.
- Wszystkie standardy oraz konwencje należy uzgodnić i udokumentować na początku projektu automatyzacji testów.

W przypadku dokumentacji należy pamiętać między innymi o następujących zasadach:

- Konieczność posiadania dobrej i aktualnej dokumentacji dotyczącej zarówno scenariuszy testowych, jak i TAS jest rzeczą oczywistą, jednak wiąże się z tym dwa problemy: potrzebny jest ktoś, kto taką dokumentację opracuje, oraz ktoś, kto będzie ją aktualizować.
- Kod narzędzia testowego może być dokumentowany automatycznie lub półautomatycznie, jednak nadal niezbędny jest ktoś, kto udokumentuje wszystkie materiały projektowe, komponenty, mechanizmy integracji z produktami innych firm, zależności oraz procedury wdrożeniowe.
- Zaleca się włączenie opracowywania dokumentacji do procesu wytwarzania oprogramowania. Oznacza to, że poszczególne zadania można uznać za ukończone dopiero po ich udokumentowaniu lub zaktualizowaniu istniejącej dokumentacji.

W przypadku materiałów szkoleniowych obowiązują między innymi następujące zasady:

- Dobrze napisana dokumentacja TAS może być podstawą do opracowania materiałów szkoleniowych dotyczących tego rozwiązania.
- Materiały szkoleniowe powinny zawierać informacje na temat specyfikacji funkcjonalnych TAS, projektu i architektury TAS, wdrożenia i utrzymania TAS oraz użytkowania TAS (instrukcja obsługi), a także praktyczne przykłady i ćwiczenia oraz porady i wskazówki.
- Utrzymanie materiałów szkoleniowych obejmuje ich opracowanie, a następnie okresową weryfikację. W praktyce zajmują się tym członkowie zespołu wyznaczeni na trenerów w dziedzinie TAS, a aktualizacja odbywa się najczęściej pod koniec danej iteracji cyklu życia systemu podlegającego testowaniu (na przykład na zakończenie każdego sprintu).

5. Raporty i metryki dotyczące automatyzacji testów — 165 min.

Słowa kluczowe

gęstość defektów w kodzie testów automatycznych, pokrycie, macierz śledzenia, ekwiwalent nakładu testowania manualnego, metryki, logowanie testu, raportowanie testów

Cele nauczania — raporty i metryki dotyczące automatyzacji testów

5.1 Wybór metryk dotyczących rozwiązania dla testów automatycznych (TAS)

ALTA-E-5.1.1 (K2) Kandydat potrafi sklasyfikować metryki, które mogą służyć do monitorowania strategii automatyzacji testów i skuteczności automatyzacji.

5.2 Implementacja pomiarów

ALTA-E-5.2.1 (K3) Kandydat potrafi zaimplementować metody zbierania metryk wspierających spełnienie wymagań technicznych i wymagań w dziedzinie zarządzania oraz wyjaśnić, w jaki sposób można zaimplementować pomiar automatyzacji testów.

5.3 Logowanie danych dotyczących rozwiązania dla testów automatycznych (TAS) i systemu podlegającego testowaniu (SUT)

ALTA-E-5.3.1 (K4) Kandydat potrafi przeanalizować sposób logowania danych związanych z testami w rozwiązaniu dla testów automatycznych i systemie podlegającym testowaniu.

5.4 Raportowanie automatyzacji testów

ALTA-E-5.4.1 (K2) Kandydat potrafi wyjaśnić, w jaki sposób należy skonstruować i opublikować raport z wykonania testów.

5.1 Wybór metryk dotyczących rozwiązania dla testów automatycznych (TAS)

W tym podrozdziale skoncentrowano się na metrykach, które mogą służyć do monitorowania strategii automatyzacji testów oraz skuteczności i efektywności rozwiązania dla testów automatycznych (TAS). Metryki te są niezależne od metryk związanych z systemem podlegającym testowaniu, które służą do monitorowania tego systemu oraz przebiegu jego testowania (funkcjonalnego i нефunkcjonalnego). Wyboru metryk dokonuje kierownik testów na szczeblu całego projektu. Metryki dotyczące automatyzacji testów umożliwiają Kierownikowi automatyzacji testów (TAM) i inżynierowi automatyzacji testów (TAE) śledzenie postępów w realizacji celów automatyzacji testów oraz monitorowanie skutków zmian wprowadzanych w TAS.

Metryki dotyczące TAS można podzielić na dwie kategorie: zewnętrzne i wewnętrzne. Metryki zewnętrzne służą do pomiaru wpływu TAS na inne działania (zwłaszcza na czynności testowe), a wewnętrzne — do pomiaru skuteczności i efektywności realizacji celów TAS.

Poniżej wymieniono typowe metryki dotyczące TAS:

Metryki zewnętrzne dotyczące TAS:

- korzyści z automatyzacji,
- nakład pracy na tworzenie testów automatycznych,
- nakład pracy na analizowanie incydentów związanych z testami automatycznymi,
- nakład pracy na pielęgnację testów automatycznych,
- stosunek liczby awarii do liczby defektów,
- czas wykonywania testów automatycznych,
- liczba zautomatyzowanych przypadków testowych,
- liczba rezultatów zaliczonych i niezaliczonych,
- liczba rezultatów fałszywie zaliczonych i fałszywie niezaliczonych,
- pokrycie kodu.

Metryki wewnętrzne dotyczące TAS:

- metryki dotyczące skryptów narzędzi,
- gęstość defektów w kodzie testów automatycznych,
- szybkość i efektywność komponentów TAS.

Poniżej opisano dokładniej poszczególne kategorie.

Korzyści wynikające z automatyzacji

Pomiar i wykazywanie korzyści wynikających z zastosowania TAS są niezwykle istotne. Wynika to z faktu, że o ile koszty (wyrażone liczbą osób zaangażowanych w projekt w danym okresie) są dobrze widoczne, przez co osoby nieuczestniczące w testowaniu łatwo mogą wyrobić sobie zdanie na temat ogólnych wydatków, o tyle uzyskane korzyści nie są już tak oczywiste.

Dobór miar służących do określania korzyści zależy od celu danego TAS. Wśród typowych korzyści można wymienić zmniejszenie czaso- lub pracochłonności testowania, zwiększenie ilości testowania (tj. szerokości/głębokości pokrycia lub częstotliwości wykonywania testów), zwiększenie powtarzalności, lepsze wykorzystanie zasobów czy zmniejszenie liczby błędów związanych z czynnościami wykonywanymi manualnie. Mierzyć można między innymi:

- zmniejszenie nakładu pracy na testowanie manualne (wyrażone liczbą godzin),
- zmniejszenie czasochłonności testowania regresji,
- liczbę dodatkowych cykli wykonywania testów,
- liczbowy lub procentowy wzrost liczby wykonywanych testów,
- udział procentowy przypadków testowych dla testów automatycznych w całym zbiorze przypadków testowych (z zastrzeżeniem, że porównywanie przypadków testowych dla testów manualnych i automatycznych nie jest łatwe),
- wzrost pokrycia (wymagań, funkcjonalności, strukturalnego),
- liczbę defektów wykrytych na wcześniejszym etapie dzięki TAS (jeśli znane są średnie korzyści wynikające z wcześniejszego wykrycia defektów, można na tej podstawie wyliczyć łączne koszty, których dzięki temu uniknięto),
- liczbę defektów wykrytych dzięki TAS, które nie zostałyby wykryte w ramach testowania manualnego (np. defektów związanych z niezawodnością).

Należy również zaznaczyć, że automatyzacja testów pozwala zwykle zmniejszyć nakłady pracy związane z testowaniem manualnym, a zwolnione w ten sposób zasoby można skierować do wykonywania innego rodzaju testów (także manualnych), na przykład do testowania eksploracyjnego. Defekty wykryte dzięki takim dodatkowym testom można również zaliczyć do korzyści pośrednich wynikających z zastosowania TAS, ponieważ ich wykonanie stało się możliwe dzięki automatyzacji testów (gdyby nie TAS, testy te nie zostałyby wykonane, a tym samym nie udałoby się wykryć dodatkowych defektów).

Nakład pracy na tworzenie testów automatycznych

Nakład pracy na tworzenie testów automatycznych jest jednym z najważniejszych składników kosztów związanych z automatyzacją testów. Koszt automatyzacji jest często wyższy niż koszt manualnego wykonywania tych samych testów, co może być przeszkodą w rozszerzaniu zakresu wykorzystania automatyzacji. O ile koszt implementacji konkretnego testu automatycznego zależy w dużej mierze od samego testu, o tyle należy również uwzględnić inne czynniki, takie jak przyjęte podejście do tworzenia skryptów, znajomość narzędzia testowego, środowisko czy poziom umiejętności inżyniera automatyzacji testów.

Zautomatyzowanie większych lub bardziej złożonych testów trwa zwykle dłużej niż zautomatyzowanie testów krótszych lub prostszych, dlatego do obliczania kosztów automatyzacji testów można wykorzystać średni czas tworzenia testu. Aby uzyskać bardziej precyzyjne wyliczenia, można dodatkowo uwzględnić średni koszt dotyczący konkretnego zbioru testów (obejmującego, na przykład, testy dotyczące tej samej funkcji lub wykonywane na tym samym poziomie testów). Innym rozwiązaniem jest wyrażenie kosztów tworzenia testów w terminach wskaźnika określającego nakład pracy niezbędny do manualnego wykonania danego testu (tzw. ekwiwalentu nakładu testowania manualnego — ang. *equivalent manual test effort*, EMTE). W ten sposób można na przykład stwierdzić, że zautomatyzowanie danego przypadku testowego jest dwa razy bardziej pracochłonne niż wykonanie go manualnie (2 EMTE).

Nakład pracy na analizowanie awarii systemu podlegającego testowaniu (SUT)

Analizowanie awarii SUT wykrytych podczas wykonywania testów automatycznych może być dużo bardziej skomplikowane niż w przypadku testów wykonywanych manualnie, ponieważ zdarzenia prowadzące do awarii w teście manualnym są zwykle znane wykonującemu go testerowi (problem ten można częściowo ograniczyć na poziomie projektowania w sposób opisany w sekcji 3.1.4 oraz na poziomie raportowania w sposób opisany w podrozdziałach 5.3 i 5.4). Ta miara w tym obszarze może być wyrażona przy użyciu średniej wartości w przeliczeniu na każdy niezaliczony przypadek testowy lub przy użyciu ekwiwalentu nakładu testowania manualnego (EMTE). To drugie rozwiązanie sprawdza się zwłaszcza w sytuacji, w której testy automatyczne różnią się znacznie złożonością i czasem wykonywania.

Niezwykle istotną rolę w analizowaniu awarii odgrywają funkcje logowania dostępne w systemie podlegającym testowaniu i TAS. Funkcje te powinny dostarczać informacji wystarczających do efektywnego przeprowadzenia tej analizy. W przypadku logowania danych ważne jest:

- synchronizowanie procesów logowania danych w systemie podlegającym testowaniu i TAS,
- logowanie przez TAS oczekiwanego i rzeczywistego zachowania,
- logowanie przez TAS czynności, które mają zostać wykonane.

Z kolei po stronie systemu podlegającego testowaniu należy logować wszystkie wykonywane czynności (bez względu na to, czy wynikają one z testowania manualnego czy automatycznego). Ponadto zaleca się logowanie wszelkich błędów wewnętrznych oraz udostępnianie ewentualnych zrzutów awaryjnych i śladów stosu (ang. *stack traces*).

Nakład pracy na pielęgnację testów automatycznych

Nakład pracy niezbędny do ciągłego synchronizowania wersji testów automatycznych z systemem podlegającym testowaniu może być bardzo duży, a w ostatecznym rozrachunku nawet przeważać nad korzyściami wynikającymi z zastosowania TAS. Było to przyczyną niepowodzenia wielu przedsięwzięć związanych z automatyzacją. Dlatego monitorowanie pracochłonności pielęgnacji jest ważnym procesem, który pozwala zasygnalizować konieczność podjęcia odpowiednich kroków w celu zmniejszenia nakładów pracy lub przynajmniej ograniczenia ich niekontrolowanego wzrostu.

Miarą nakładu pracy związanego z pielęgnacją może być łączna pracochłonność pielęgnacji wszystkich testów automatycznych, które wymagają aktualizacji w związku z każdą nową wersją systemu podlegającego testowaniu. Oprócz tego można posłużyć się średnią wartością w przeliczeniu na każdy zaktualizowany test automatyczny lub wyrazić ją jako część EMTE.

Metryką pokrewną jest liczba lub odsetek testów, w przypadku których niezbędne jest wykonanie prac związanych z pielęgnacją.

Wiedza na temat pracochłonności pielęgnacji testów automatycznych (lub możliwość wyznaczenia tej wielkości w sposób pośredni) może odegrać istotną rolę przy podejmowaniu decyzji o zaimplementowaniu/niezaimplementowaniu określonych funkcjonalności bądź o usunięciu/nieusunięciu określonego defektu. Nakład pracy niezbędny do zaktualizowania przypadku testowego w związku z modyfikacją oprogramowania należy rozpatrywać łącznie z modyfikacją SUT.

Stosunek liczby awarii do liczby defektów

Powszechnym problemem związanym z testami automatycznymi jest fakt, że wiele z nich kończy się niepowodzeniem z tego samego powodu: na skutek pojedynczego defektu w oprogramowaniu. Testy mają oczywiście na celu wskazanie defektów oprogramowania, ale wykrywanie tego samego defektu przez kilka testów jest nieekonomiczne — zwłaszcza w przypadku testów automatycznych, gdzie nakład pracy związany z analizą każdego niezaliczonego testu może być znaczny. Pomiar liczby testów automatycznych, które kończą się niepowodzeniem na skutek określonego defektu, może pomóc zidentyfikować sytuacje, w których może wystąpić tego rodzaju problem. Rozwiązaniem jest wówczas właściwe zaprojektowanie testów automatycznych i właściwy dobór wykonywanych testów.

Czas wykonywania testów automatycznych

Jedną z łatwiejszych do określenia metryk jest czas potrzebny do wykonania testów automatycznych. Na początkowym etapie eksploatacji TAS może to być mniej istotne, ale wraz ze wzrostem liczby zautomatyzowanych przypadków testowych metryka ta może nabrać stosunkowo dużego znaczenia.

Liczba zautomatyzowanych przypadków testowych

Ta metryka może posłużyć do określenia postępu realizacji projektu automatyzacji testów. Należy jednak wziąć pod uwagę to, że sama liczba zautomatyzowanych przypadków testowych nie dostarcza zbyt wielu informacji (na przykład, nie wskazuje, czy wzrosło pokrycie testowe).

Liczba rezultatów zaliczonych i niezaliczonych

Jest to powszechnie stosowana metryka, która służy do śledzenia liczby zaliczonych testów automatycznych oraz testów, w przypadku których nie osiągnięto oczekiwanego rezultatu. Każdy rezultat niezaliczony należy przeanalizować, aby ustalić, czy test nie został zaliczony na skutek defektu w systemie podlegającym testowaniu, czy też w wyniku problemów zewnętrznych, takich jak nieprawidłowe działanie środowiska lub samego TAS.

Liczba rezultatów fałszywie zaliczonych i fałszywie niezaliczonych

Jak już wspomniano przy kilku poprzednich metrykach, analizowanie niezaliczonych testów może być dość czasochłonne. Sytuacja jest tym bardziej frustrująca, gdy okazuje się, że mamy do czynienia z fałszywym alarmem. Dzieje się tak, jeśli problem leży po stronie TAS lub przypadku testowego, a nie po stronie systemu podlegającego testowaniu. Nie ulega zatem wątpliwości, że liczbę fałszywych alarmów (wiązących się potencjalnie ze zbędnymi nakładami pracy) należy utrzymywać na jak najniższym poziomie. Rezultaty fałszywie niezaliczone mogą powodować spadek zaufania do TAS, ale to rezultaty fałszywie zaliczone mogą okazać się groźniejsze. Rezultat fałszywie zaliczony oznacza, że w systemie podlegającym testowaniu wystąpiła awaria, ale nie została ona zidentyfikowana przez mechanizm automatyzacji testów, przez co zgłoszony został rezultat pozytywny. W takim przypadku potencjalny defekt może pozostać niewykryty. Przyczyną może być nieprawidłowa weryfikacja wyniku, zastosowanie niepoprawnej wyroczni testowej lub nieprawidłowe określenie oczekiwanego rezultatu w przypadku testowym.

Należy pamiętać, że fałszywe alarmy mogą być spowodowane zarówno defektami w kodzie testowym (patrz metryka „gęstość defektów w kodzie testów automatycznych”), jak i nieprzewidywalnym zachowaniem niestabilnego systemu podlegającego testowaniu (np. przekroczeniem limitu czasu). Z uwagi na wysoki poziom inwazyjności przyczyną fałszywych alarmów mogą być również zaczepy testowe (ang. *test hooks*).

Pokrycie kodu

Określenie pokrycia kodu systemu podlegającego testowaniu przez poszczególne przypadki testowe pozwala uzyskać przydatne informacje. Parametr ten można również mierzyć na wysokim poziomie, na przykład w kategoriach pokrycia kodu przez zestaw testów regresji. Nie istnieje przy tym jedna wartość procentowa wskazująca na wystarczające pokrycie, a pokrycie na poziomie 100% kodu jest osiągalne jedynie w najprostszych aplikacjach. Powszechnie uznaje się jednak, że większe pokrycie jest korzystne, ponieważ ogranicza ogólne ryzyko związane z wdrożeniem oprogramowania. Metryka ta może również sygnalizować

wykonanie określonych czynności w systemie podlegającym testowaniu — na przykład spadek pokrycia kodu oznacza, że do systemu dodano najprawdopodobniej kolejne funkcje, a do zestawu testowego testów automatycznych nie dodano jeszcze odpowiedniego przypadku testowego.

Metryki dotyczące skryptów narzędzi

Istnieje wiele metryk umożliwiających monitorowanie procesu tworzenia skryptów automatyzacji, przy czym w większości są one podobne do metryk stosowanych w odniesieniu do kodu źródłowego systemu podlegającego testowaniu. Na przykład, liczba linii kodu (ang. *line of code* – LOC) i złożoność cyklomatyczna pozwalają zidentyfikować nadmiernie rozbudowane lub skomplikowane skrypty (sugerując możliwe przeprojektowanie, o ile okaże się to potrzebne).

Stosunek liczby komentarzy do liczby instrukcji wykonywalnych może pomóc w określeniu zakresu niezbędnej dokumentacji i adnotacji skryptu, a liczba niezgodności ze standardami dotyczącymi tworzenia skryptów wskazuje, na ile przestrzegane są obowiązujące zasady.

Gęstość defektów w kodzie testów automatycznych

Podobnie jak kod systemu podlegającego testowaniu, kod testów automatycznych również stanowi oprogramowanie i może zawierać defekty, w związku z czym nie powinien być traktowany jako mniej ważny niż kod systemu podlegającego testowaniu. Oznacza to, że również w tym przypadku należy stosować dobre praktyki i standardy dotyczące tworzenia kodu oraz monitorować rezultaty przy użyciu odpowiednich metryk, takich jak gęstość defektów w kodzie testów automatycznych. Zbieranie metryk jest łatwiejsze w przypadku korzystania z systemu zarządzania konfiguracją.

Szybkość i efektywność komponentów TAS

Różne czasy wykonania tych samych kroków testów w tym samym środowisku mogą sygnalizować problem po stronie systemu podlegającego testowaniu. Jeśli SUT nie wykonuje tych samych funkcji w tym samym czasie, należy podjąć działania wyjaśniające. Objawy te mogą wskazywać na zmienność parametrów systemu, która jest nie do przyjęcia i może jeszcze pogłębić się pod zwiększonym obciążeniem. Trzeba jednocześnie zaznaczyć, że wydajność TAS musi być na tyle duża, aby działanie rozwiązania nie wpływało negatywnie na wydajność systemu podlegającego testowaniu. Jeśli wydajność jest krytycznym wymaganiem stawianym systemowi podlegającemu testowaniu, należy to uwzględnić przy projektowaniu TAS.

Metryki dotyczące trendów

W przypadku wielu opisanych powyżej metryk cenniejsze od konkretnych wartości chwilowych mogą być zaobserwowane trendy, czyli sposób, w jaki wartości te zmieniają się w czasie. Na przykład informacja o tym, że średni koszt utrzymania w przeliczeniu na każdy test automatyczny wymagający aktualizacji jest wyższy niż w przypadku poprzednich dwóch wersji systemu podlegającego testowaniu, pozwala podjąć działania mające na celu ustalenie przyczyn wzrostu kosztów i odwrócenie niekorzystnego trendu.

Koszt pomiaru powinien być jak najniższy, co często można osiągnąć poprzez zautomatyzowanie procesów zbierania danych i raportowania.

5.2 Implementacja pomiarów

Z uwagi na to, że testalia do testów automatycznych stanowią podstawę strategii testów automatycznych, ich funkcjonalność można rozszerzyć o rejestrowanie informacji na temat stosowania tej strategii. Co więcej, dzięki połączeniu abstrakcji z ustrukturyzowanymi testaliami wszelkie udoskonalenia wprowadzane w bazowych testalich mogą być wykorzystywane przez wszystkie skrypty testów automatycznych wyższego poziomu. Przykładem takiego udoskonalenia mającego potencjalnie zastosowanie do wszystkich testów jest wprowadzenie w bazowych testalich funkcji rejestrowania czasu rozpoczęcia i zakończenia testu.

Funkcje automatyzacji wspomagające dokonywanie pomiarów i generowanie raportów

Aby ułatwić dokonywanie pomiarów i raportowanie, języki skryptowe używane w wielu narzędziach testowych oferują mechanizmy umożliwiające zapisywanie i rejestrowanie informacji przed wykonaniem, podczas wykonywania i po wykonaniu pojedynczych testów, zbiorów testów lub całego zestawu testowego.

Narzędzia służące do raportowania na temat poszczególnych przebiegów testów w serii muszą być wyposażone w funkcję analizy, która uwzględnia rezultaty poprzednich przebiegów testów i pozwala określić na tej podstawie trendy związane na przykład ze zmianą wskaźnika powodzenia testów.

Automatyzacja testów wymaga zwykle zautomatyzowania zarówno wykonywania testów, jak i weryfikacji wyników tych testów, przy czym w tym drugim przypadku odbywa się to poprzez porównywanie określonych elementów wyniku testu ze zdefiniowanym wcześniej oczekiwanym rezultatem. Do przeprowadzania tego typu porównań najlepiej nadaje się narzędzie testowe (należy wziąć pod uwagę poziom informacji raportowanych w wyniku porównania). Kolejną ważną kwestią jest prawidłowe określenie statusu testu (np. zaliczony lub niezaliczony), ponieważ w przypadku niezaliczenia testu wymagane będą dodatkowe informacje na temat przyczyny niepowodzenia (np. rzuty ekranu).

Rozróżnienie pomiędzy spodziewanymi różnicami między rzeczywistym a oczekiwanym wynikiem testu nie zawsze jest rzeczą prostą, ale narzędzia mogą bardzo pomóc w zdefiniowaniu porównań, które ignorują spodziewane różnice (dotyczące np. daty i godziny), a wyróżniają różnice nieoczekiwane.

Integracja z narzędziami innych firm (arkuszami kalkulacyjnymi, plikami XML, dokumentami, bazami danych, narzędziami do raportowania itd.)

Jeśli informacje uzyskane podczas wykonywania zautomatyzowanych przypadków testowych mają być wykorzystywane przez inne narzędzia (do celów związanych ze śledzeniem i raportowaniem, np. do aktualizowania macierzy śledzenia), można je udostępnić w formacie odpowiednim dla narzędzi innych firm. Często wykorzystuje się do tego istniejące funkcje narzędzi (formaty eksportu do raportowania) lub tworzy się niestandardowe mechanizmy raportowania, które generują dane wyjściowe w formacie dostosowanym do potrzeb innych programów („xls” w przypadku programu Excel, „doc” w przypadku programu Word, „html” w przypadku przeglądarek internetowych itd.).

Wizualizacja rezultatów

Rezultaty testów powinny być prezentowane w formie wykresów. Warto przy tym zastanowić się nad użyciem różnych kolorów do sygnalizowania problemów związanych z wykonywaniem testów, na przykład sygnalizatorów świetlnych wskazujących postęp wykonywania/automatyzacji testów, ponieważ ułatwi to podejmowanie decyzji na podstawie informacji zawartych w raportach. Menedżerowie szczególnie cenią sobie podsumowania wizualne, które pozwalają na pierwszy rzut oka ocenić rezultaty testów (jeśli zajdzie potrzeba, zawsze można przejść do bardziej szczegółowych informacji).

5.3 Logowanie danych dotyczących rozwiązania dla testów automatycznych (TAS) i systemu podlegającego testowaniu (SUT)

Bardzo ważnym aspektem rozwiązania dla testów automatycznych (TAS) jest logowanie danych dotyczących zarówno samej automatyzacji testów, jak i systemu podlegającego testowaniu. Dzienniki (logi) testów są źródłem informacji, do którego często sięga się w celu przeanalizowania potencjalnych problemów. Poniżej przedstawiono przykłady informacji rejestrowanych w związku z testami — osobno w odniesieniu do TAS i systemu podlegającego testowaniu.

Logowanie po stronie TAS (to, czy informacje są rejestrowane na poziomie TAF, czy też na poziomie przypadków testowych, nie jest szczególnie istotne i zależy od kontekstu) powinno obejmować następujące elementy:

- Należy logować informacje o tym, który przypadek testowy jest aktualnie wykonywany (z uwzględnieniem czasu rozpoczęcia i zakończenia).
- Należy logować status wykonania przypadku testowego. Niezaliczone testy można łatwo zidentyfikować w dziennikach testów, ale informacje na ten temat powinny być również dostępne na poziomie struktury do testów automatycznych (TAF) i raportowane za pośrednictwem tablicy wskaźników (ang. *dashboard*). Status wykonania przypadku testowego może mieć wartość „zaliczony”, „niezaliczony” lub „błąd TAS”, przy czym ta ostatnia wartość jest stosowana, jeśli przyczyna problemu nie leży po stronie systemu podlegającego testowaniu.
- Szczegóły dziennika testów, w tym informacje o czasach wykonania, należy rejestrować na wysokim poziomie (rejestrowanie istotnych kroków).
- Należy logować informacje dynamiczne na temat systemu podlegającego testowaniu (dotyczące np. wycieków pamięci), które udało się zidentyfikować w ramach przypadku testowego z pomocą

- narzędzi innych firm. Rzeczywiste rezultaty i niepowodzenia pomiarów dynamicznych należy rejestrować razem z przypadkiem testowym, który był wykonywany w momencie wykrycia incydentu.
- W przypadku testowania niezawodności lub testowania przeciążającego (w ramach którego wykonywanych jest wiele cykli) należy logować wartość licznika, co pozwoli łatwo ustalić, ile razy zostały wykonane przypadki testowe.
 - Jeśli przypadki testowe zawierają elementy losowe (np. parametry losowe lub kroki losowe w przypadku testowania z użyciem maszyny stanów), należy logować wartości takich liczb/parametrów losowych.
 - Wszystkie akcje wykonywane w ramach przypadku testowego należy logować w taki sposób, aby było możliwe odtworzenie pliku dziennika (lub jego fragmentów) w celu ponownego wykonania testu z użyciem tych samych kroków i z zachowaniem tych samych parametrów czasowych. Pozwoli to sprawdzić, czy istnieje możliwość odtworzenia zidentyfikowanej awarii, a także pozyskać dodatkowe informacje. Informacje o akcjach wykonywanych w ramach przypadku testowego można również rejestrować w systemie podlegającym testowaniu i wykorzystywać do odtwarzania problemów zidentyfikowanych przez klienta (w takim przypadku klient uruchamia scenariusz, a w dzienniku są zapisywane informacje, które zespół programistyczny może następnie odtworzyć podczas diagnozowania problemu).
 - Podczas wykonywania testów można zapisywać zrzuty ekranu i inne informacje wizualne przeznaczone do dalszego wykorzystania na etapie analizy awarii.
 - Rozwiązanie dla testów automatycznych powinno zagwarantować, że w momencie wykrycia przez przypadek testowy każdej awarii zostaną udostępnione lub zapisane wszelkie informacje niezbędne do przeanalizowania problemu oraz ewentualne informacje dotyczące kontynuacji testowania. Ponadto, wskazane jest zapisywanie przez TAS w bezpiecznym miejscu wszelkich związanych z tym zrzutów awaryjnych i śladów stosu. Z myślą o późniejszej analizie, w to samo miejsce powinny być również kopiowane pliki dzienników, które mogłyby zostać nadpisane (do obsługi plików dzienników po stronie systemu podlegającego testowaniu są często używane bufora cykliczne).
 - Poszczególne typy rejestrowanych informacji można wyróżniać kolorami (np. błędy kolorem czerwonym, a informacje o postępie - kolorem zielonym).

Logowanie po stronie systemu podlegającego testowaniu:

- Gdy SUT zidentyfikuje problem, należy zarejestrować wszelkie informacje niezbędne do jego przeanalizowania, w tym znaczniki daty i godziny, informacje o źródle problemu, komunikaty o błędach itd.
- System podlegający testowaniu może logować całość interakcji z użytkownikami (zarówno bezpośrednio przy użyciu dostępnego interfejsu użytkownika, jak i za pośrednictwem interfejsów sieciowych itd.). Dzięki temu problemy zidentyfikowane przez klientów mogą zostać prawidłowo przeanalizowane, a programiści mogą podjąć próbę ich odtworzenia.
- W momencie uruchomienia systemu należy zarejestrować w pliku informacje konfiguracyjne, w tym informacje o poszczególnych wersjach oprogramowania (łącznie z oprogramowaniem układowym), konfiguracji SUT, konfiguracji systemu operacyjnego itd.

Należy również zadbać o możliwość łatwego przeszukiwania wszystkich zarejestrowanych informacji. Problem zidentyfikowany w pliku dziennika po stronie TAS powinien być łatwy do zidentyfikowania w pliku dziennika systemu i odwrotnie (przy użyciu dodatkowych narzędzi lub bez ich użycia). Ponadto ważna jest możliwość synchronizacji poszczególnych dzienników przy użyciu znaczników czasu, co ułatwia korelację zdarzeń, które miały miejsce w momencie zgłoszenia błędu.

5.4 Raportowanie wyników testów automatycznych

Dzienniki (logi) testów dostarczają szczegółowych informacji na temat kroków testów oraz akcji i odpowiedzi związanych z przypadkiem testowym i/lub zestawem testów. Na podstawie samych dzienników nie można jednak uzyskać szerokiego obrazu ogólnych rezultatów wykonania testów, ponieważ do tego potrzebne są funkcje raportowania. Po każdym wykonaniu zestawu testowego musi zostać utworzony i opublikowany zwięzły raport. Można w tym celu wykorzystać komponent generatora raportów wielokrotnego użytku.

Treść raportów

Raport z wykonania testów musi zawierać podsumowanie dostarczające podstawowych informacji na temat rezultatów, testowanego systemu oraz środowiska, w którym zostały wykonane testy, w formie odpowiedniej dla poszczególnych interesariuszy.

Niezbędna jest informacja o tym, które testy nie zostały zaliczone i jaka była tego przyczyna. Aby ułatwić diagnozowanie problemów, należy również uwzględnić informacje o historii wykonywania danego testu oraz o osobie odpowiedzialnej za ten test (zwykle jest to osoba, która utworzyła lub ostatnio zaktualizowała test). Osoba odpowiedzialna musi zbadać przyczynę niezaliczenia testu, zgłosić związane z nią problemy, dopilnować rozwiązania problemów oraz sprawdzić, czy rozwiązanie zostało prawidłowo zaimplementowane.

Raportowanie umożliwia również diagnozowanie ewentualnych awarii komponentów struktury do testów automatycznych (TAF) (patrz Rozdział 7.).

Publikowanie raportów

Raport należy opublikować i udostępnić wszystkim osobom zainteresowanym rezultatami wykonania testów. Można go umieścić na stronie internetowej, wysłać do osób z listy adresowej lub przesłać do innego narzędzia, na przykład narzędzia do zarządzania testami. Z praktycznego punktu widzenia, najlepszą gwarancją, że osoby zainteresowane rezultatami testów zapoznają się z raportem i przeanalizują jego treść, daje udostępnienie tym osobom mechanizmu subskrypcji i wysłanie im raportu pocztą elektroniczną.

Inną opcją ułatwiającą identyfikowanie problematycznych elementów systemu podlegającego testowaniu jest przechowywanie raportów historycznych i gromadzenie danych statystycznych na temat tych przypadków testowych lub zestawów testowych, które są związane z częstą regresją.

6. Przeniesienie testowania manualnego do środowiska zautomatyzowanego — 120 min.

Słowa kluczowe

testowanie potwierdzające, testowanie regresji

Cele nauczania — przeniesienie testowania manualnego do środowiska zautomatyzowanego

6.1 Kryteria automatyzacji

- ALTA-E-6.1.1 (K3) Kandydat potrafi stosować kryteria służące do określania testów nadających się do zautomatyzowania.
- ALTA-E-6.1.2 (K2) Kandydat zna czynniki związane z przejściem z testowania manualnego na automatyczne.

6.2 Określenie kroków niezbędnych do implementacji zautomatyzowanych testów regresji

- ALTA-E-6.2.1 (K2) Kandydat potrafi wyjaśnić, jakie czynniki należy wziąć pod uwagę przy automatyzowaniu testów regresji.

6.3 Czynniki, które należy wziąć pod uwagę przy implementacji testów automatycznych nowych funkcjonalności (cech)

- ALTA-E-6.3.1 (K2) Kandydat potrafi wyjaśnić, jakie czynniki należy wziąć pod uwagę przy automatyzowaniu nowych funkcjonalności (cech).

6.4 Czynniki, które należy wziąć pod uwagę przy implementacji automatycznych testów potwierdzających

- ALTA-E-6.4.1 (K2) Kandydat potrafi wyjaśnić, jakie czynniki należy wziąć pod uwagę przy automatyzowaniu testów potwierdzających.

6.1 Kryteria automatyzacji

Organizacje od dawna tworzą przypadki testowe dla testów manualnych, dlatego przy podejmowaniu decyzji o migracji do zautomatyzowanego środowiska testowego należy ocenić bieżący stan testów manualnych i określić najskuteczniejszy sposób automatyzacji istniejących zasobów. Dotychczasowa struktura poszczególnych testów manualnych może, ale nie musi, nadawać się do zautomatyzowania, w związku z czym może być konieczne napisanie niektórych testów od nowa pod kątem automatyzacji. Innym rozwiązaniem jest wyodrębnienie z istniejących testów manualnych odpowiednich komponentów (np. wartości wejściowych, oczekiwanych rezultatów czy ścieżki nawigacji) i ponowne wykorzystanie ich do automatyzacji. Korzystne może być również przyjęcie strategii testów manualnych, która uwzględnia potencjalną automatyzację i umożliwia tworzenie testów o strukturze ułatwiającej migrację do środowiska zautomatyzowanego.

Nie wszystkie testy można lub należy automatyzować, a nawet w przypadku automatyzacji pierwsza iteracja może być nadal wykonywana manualnie. W związku z tym należy wziąć pod uwagę dwa aspekty związane ze zmianą sposobu testowania: początkowe przekształcanie istniejących testów manualnych w automatyczne oraz późniejsze przekształcanie nowych testów manualnych w automatyczne.

Trzeba również zaznaczyć, że niektóre typy testów mogą być (skutecznie) wykonywane tylko w sposób automatyczny — dotyczy to na przykład testów niezawodności, testów przeciążających czy testów wydajnościowych.

Testy automatyczne umożliwiają testowanie aplikacji i systemów, które nie mają interfejsu użytkownika. W takim przypadku testy można wykonywać na poziomie integracji za pośrednictwem interfejsów dostępnych w oprogramowaniu. Teoretycznie powyższe przypadki testowe można by było również wykonywać manualnie (przy użyciu ręcznie wprowadzanych poleceń służących do wywoływania interfejsów), ale rozwiązanie takie jest w wielu przypadkach niepraktyczne. Dzięki automatyzacji można na przykład umieszczać komunikaty w systemie zarządzającym kolejką komunikatów, aby rozpocząć testowanie (i zidentyfikować ewentualne defekty) na wcześniejszym etapie, na którym testowanie manualne nie jest jeszcze możliwe.

Przed rozpoczęciem prac związanych z automatyzacją testowania należy zastanowić się nad zasadnością i opłacalnością tworzenia testów automatycznych zamiast manualnych. Do kryteriów służących do określania zasadności automatyzacji zaliczają się w szczególności:

- częstotliwość używania,
- złożoność automatyzacji,
- kompatybilność i obsługa narzędzi,
- dojrzałość procesu testowego,
- zasadność automatyzacji na danym etapie cyklu życia oprogramowania,
- trwałość zautomatyzowanego środowiska,
- sterowalność systemu podlegającego testowaniu.

Poniżej opisano dokładniej poszczególne kryteria.

Częstotliwość używania

Jednym z czynników decydujących o zasadności ewentualnej automatyzacji jest częstotliwość, z jaką musi być wykonywany dany test. Do zautomatyzowania lepiej nadają się testy wykonywane częściej i bardziej regularnie, na przykład w ramach cyklu wersji głównych lub podwersji. Co do zasady, im więcej wersji ma aplikacja (a tym samym odpowiadających im cykli testowych), tym większe korzyści przynosi automatyzacja testów. Zautomatyzowane testy funkcjonalne mogą być następnie stosowane w kolejnych wersjach produktu w ramach testowania regresji. Zastosowanie testów automatycznych do testowania regresji zapewnia duży zwrot z inwestycji i pozwala skutecznie złagodzić ryzyko dotyczące dotychczasowego kodu.

Jeśli skrypt testowy jest wykonywany tylko raz w roku, a cykl wprowadzania zmian w systemie podlegającym testowaniu jest podobny, automatyzacja takiego testu może okazać się niemożliwa lub nieefektywna. Biorąc pod uwagę czas, jaki należałoby poświęcić co roku na dostosowanie testu do zmian w systemie, lepszym rozwiązaniem może być wykonanie takiego testu manualnie.

Złożoność automatyzacji

Jeśli konieczne jest przetestowanie złożonego systemu, automatyzacja może przynieść ogromne korzyści, ponieważ dzięki niej testerzy nie muszą manualnie powtarzać złożonych kroków, których wykonywanie jest żmudne, czasochłonne i obciążone ryzykiem błędów.

Z drugiej strony, w przypadku niektórych skryptów testowych, automatyzacja może być trudna lub nieopłacalna. Przyczyn takiej sytuacji może być wiele: brak kompatybilności systemu podlegającego testowaniu z istniejącymi rozwiązaniami do testowania automatycznego; konieczność opracowania dużej ilości kodu programu bądź wywołań interfejsów API w celu zautomatyzowania testów; duża liczba systemów wymagających uwzględnienia przy wykonywaniu testów; interakcja z zewnętrznymi interfejsami i/lub niestandardowymi systemami; określone aspekty testowania użyteczności; czas potrzebny na przetestowanie skryptów testów automatycznych itd.

Kompatybilność i obsługa narzędzi

Do tworzenia aplikacji używa się szerokiej gamy platform programistycznych. W tej sytuacji tester musi wiedzieć, czy i jakie narzędzia testowe są dostępne w przypadku danej platformy oraz w jakim stopniu jest ona wspierana. Organizacje korzystają z wielu różnych narzędzi do testowania, w tym gotowych narzędzi komercyjnych, narzędzi *open source* oraz narzędzi wytwarzanych we własnym zakresie. Ponadto każda organizacja ma inne potrzeby i możliwości w zakresie obsługi takich narzędzi. Dostawcy komercyjnych narzędzi testowych zwykle oferują płatną pomoc techniczną, a liderzy rynku dysponują również siecią ekspertów mogących udzielić pomocy przy implementacji tych narzędzi. W przypadku narzędzi *open source* pomoc może być dostępna na forach dyskusyjnych, na których użytkownicy mogą uzyskiwać informacje i zadawać pytania, natomiast w przypadku narzędzi wytworzonych we własnym zakresie zapewnienie pomocy technicznej jest zadaniem pracowników firmy.

Nie należy lekceważyć problemów związanych z kompatybilnością narzędzi testowych. Rozpoczęcie projektu automatyzacji testów bez pełnej wiedzy na temat poziomu zgodności narzędzi testowych z systemem podlegającym testowaniu może mieć fatalne konsekwencje. Nawet jeśli większość testów wykonywanych w systemie podlegającym testowaniu da się zautomatyzować, może się okazać, że jest to niemożliwe w przypadku testów uznawanych za najbardziej newralgiczne.

Dojrzałość procesu testowego

Aby można było skutecznie zaimplementować testy automatyczne w procesie testowym, proces ten musi być ustrukturyzowany, uporządkowany i powtarzalny. Automatyzacja powoduje włączenie całego procesu wytwarzania oprogramowania w dotychczasowy proces testowy, co wiąże się z koniecznością zarządzania kodem testów automatycznych i związanymi z nim komponentami.

Zasadność automatyzacji na danym etapie cyklu życia oprogramowania

System podlegający testowaniu ma określony cykl życia, który może obejmować od kilku do kilkudziesięciu lat. Od momentu rozpoczęcia prac programistycznych system jest stale modyfikowany i rozszerzany w celu usuwania defektów oraz wprowadzania ulepszeń zgodnych z potrzebami docelowych użytkowników. Na wczesnych etapach rozwoju systemu zmiany mogą zachodzić zbyt szybko, aby można było zaimplementować zautomatyzowane rozwiązanie do testowania. W dynamicznie zmieniającym się środowisku, w którym trwa optymalizowanie i udoskonalanie układu ekranów czy elementów sterujących, tworzenie mechanizmów automatyzacji wymagałoby ciągłych przeróbek, przez co byłoby nieskuteczne i nieefektywne. Można to porównać do próby wymiany opony w jadącym samochodzie — w takiej sytuacji lepiej jest poczekać, aż pojazd się zatrzyma. W przypadku dużych systemów wytwarzanych metodą sekwencyjną implementację testów automatycznych najlepiej jest zacząć w momencie, gdy system zostanie już ustabilizowany i będzie zawierał podstawowe funkcje.

Z czasem każdy system osiąga kres swojego cyklu życia i jest wycofywany lub przeprojektowywany pod kątem zastosowania nowszych, bardziej efektywnych technologii. Wprowadzanie automatyzacji w systemie znajdującym się w końcowej fazie cyklu życia nie jest zalecane, ponieważ realizacja takiej krótkotrwałej inicjatywy nie przyniosłaby istotnych korzyści. Z drugiej strony, w przypadku systemów, które są przeprojektowywane pod kątem zastosowania innej architektury, ale z zachowaniem dotychczasowej funkcjonalności, zautomatyzowane środowisko testowania definiujące elementy danych może być tak samo przydatne w starej i w nowej wersji systemu. W tej sytuacji istnieje możliwość ponownego wykorzystania danych testowych, pod warunkiem zmodyfikowania kodu zautomatyzowanego środowiska w celu zapewnienia kompatybilności z nową architekturą.

Trwałość zautomatyzowanego środowiska

Środowisko testowe tworzone na potrzeby automatyzacji musi charakteryzować się elastycznością oraz możliwością adaptacji do zmian zachodzących z czasem w systemie podlegającym testowaniu. W tym zakresie ważne są między innymi możliwość szybkiego diagnozowania i rozwiązywania problemów z automatyzacją, łatwość utrzymania komponentów testów automatycznych oraz łatwość dodawania nowych funkcji i mechanizmów pomocniczych w zautomatyzowanym środowisku. Atrybuty te są nieodłącznym elementem ogólnego procesu projektowania i implementacji gTAA.

Sterowalność systemu podlegającego testowaniu (warunki wstępne, konfiguracja i stabilność)

Inżynier automatyzacji testów (TAE) powinien zidentyfikować parametry systemu podlegającego testowaniu związane ze sterowaniem i widocznością, które będą pomocne przy tworzeniu skutecznych testów automatycznych. W przeciwnym razie automatyzacja testów będzie opierać się wyłącznie na interakcjach za pośrednictwem interfejsu użytkownika, co utrudni utrzymanie rozwiązania dla testów automatycznych. Więcej informacji na ten temat zawiera podrozdział 2.3 (Projektowanie pod kątem testowalności i automatyzacji).

Planowanie techniczne na potrzeby analizy zwrotu z inwestycji

Korzyści, jakie przynosi zespołowi testowemu automatyzacja testów, zależą od konkretnej sytuacji. Jednocześnie implementacja skutecznego zautomatyzowanego rozwiązania do testowania wiąże się ze znacznymi nakładami pracy i kosztami. Dlatego przed rozpoczęciem czaso- i pracochłonnego procesu tworzenia testów automatycznych należy oszacować zakładane i potencjalne łączne korzyści wynikające z implementacji mechanizmów automatyzacji testowania, a także skutki takiej implementacji. Po ustaleniu powyższych parametrów należy określić działania niezbędne do wprowadzenia w życie przyjętego planu i związane z nimi koszty, aby obliczyć zwrot z inwestycji. Aby należycie przygotować się do przejścia na środowisko zautomatyzowane, należy uwzględnić:

- dostępność narzędzi umożliwiających automatyzację testów w środowisku testowym,
- poprawność danych testowych i przypadków testowych,
- zakres prac związanych z automatyzacją testów,
- uświadomienie zespołowi testowemu skutków zmiany paradygmatu,
- role i odpowiedzialności,
- współpracę między programistami a inżynierami automatyzacji testów,
- równoległe wykonywanie prac,
- raportowanie na temat automatyzacji testów.

Dostępność narzędzi umożliwiających automatyzację testów w środowisku testowym

W pierwszej kolejności należy zainstalować wybrane narzędzia testowe w środowisku testowym (laboratorium) i potwierdzić ich prawidłowe funkcjonowanie. Może to wymagać pobrania ewentualnych pakietów serwisowych lub aktualizacji, wybrania odpowiedniej konfiguracji instalacji (w tym dodatków) niezbędnej do obsługi systemu podlegającego testowaniu oraz sprawdzenia, czy TAS funkcjonuje poprawnie w środowisku testowym oraz w środowisku, w którym są prowadzone prace nad automatyzacją.

Poprawność danych testowych i przypadków testowych

Poprawność i kompletność danych i przypadków testowych używanych do wykonywania testów manualnych jest warunkiem uzyskania przewidywalnych rezultatów przy zastosowaniu automatyzacji. Testy wykonywane w sposób zautomatyzowany wymagają jednoznacznych danych wejściowych, nawigacyjnych, synchronizacyjnych i walidacyjnych.

Zakres prac związanych z automatyzacją testów

Aby szybko wykazać korzyści wynikające z automatyzacji, należy uzyskać informacje zwrotne na temat problemów technicznych mogących wpłynąć na postęp prac oraz ułatwić wykonywanie dalszych zadań związanych z automatyzacją, należy ograniczyć początkowy zakres projektu. Projekt pilotażowy może skupić się na jednym obszarze funkcjonalności systemu, który jest reprezentatywny dla współdziałania całego systemu. Doświadczenie zdobyte podczas pilotażu pomoże następnie skorygować oszacowania czasu i harmonogramy oraz wskazać obszary wymagające zastosowania wyspecjalizowanych zasobów technicznych. W ten sposób projekt pilotażowy pozwoli szybko wykazać skuteczność automatyzacji, a w rezultacie uzyskać większe wsparcie ze strony kierownictwa.

W osiągnięciu powyższych celów pomaga rozsądny wybór przypadków testowych, które mają zostać zautomatyzowane. Należy wybrać przypadki, które dadzą się zautomatyzować niewielkim nakładem pracy, a przy tym pozwolą uzyskać duże korzyści. Przykładem mogą być automatyczne testy regresji lub dymne — ich automatyzacja może być bardzo opłacalna, ponieważ są zwykle wykonywane dość często, niekiedy nawet codziennie. Innym dobrym punktem wyjścia jest testowanie niezawodności. Tego rodzaju testy składają się

często z wielu kroków i są wykonywane wielokrotnie w celu ujawnienia problemów, które są trudne do wykrycia metodą manualną. Implementacja testów niezawodności jest mało pracochłonna, a korzyści mogą być widoczne bardzo szybko.

Projekty pilotażowe pozwalają skierować uwagę na zalety automatyzacji (takie jak zmniejszenie nakładów pracy manualnej czy możliwość zidentyfikowania poważnych problemów) i utorować drogę dla dalszych rozszerzeń (wymagających nakładów pracy i nakładów finansowych).

Ważne jest również nadanie priorytetu testom, które mają krytyczne znaczenie dla organizacji, ponieważ w ich przypadku można wykazać na początku największe korzyści. W tym kontekście należy jednak pamiętać, że na etapie pilotażu lepiej jest nie uwzględniać testów, które są najtrudniejsze do zautomatyzowania z technicznego punktu widzenia. W przeciwnym razie próby automatyzacji mogą okazać się zbyt pracochłonne w stosunku do osiągniętych rezultatów. Co do zasady, zidentyfikowanie testów, które mają cechy wspólne z dużą częścią aplikacji, pozwoli uzyskać dynamikę niezbędną do kontynuowania wysiłków na rzecz automatyzacji.

Uświadomienie zespołowi testowemu skutków zmiany paradygmatu

Testerzy nie tworzą jednorodnej grupy: niektórzy z nich są ekspertami merytorycznymi (w zakresie danej domeny) wywodzącymi się ze społeczności użytkowników końcowych lub analityków biznesowych, inni zaś mają duże umiejętności techniczne, dzięki którym lepiej orientują się w istniejącej architekturze systemu. Zróżnicowanie to jest bardzo korzystne z punktu widzenia skuteczności testowania, ale należy też pamiętać o tym, że w miarę przechodzenia na zautomatyzowany tryb pracy poszczególne role stają się coraz bardziej wyspecjalizowane. Zmiany składu zespołu testowego mają zasadnicze znaczenie dla powodzenia automatyzacji, dlatego należy jak najwcześniej uświadomić członkom zespołu konieczność dokonywania takich zmian, aby ograniczyć ewentualne obawy związane z podziałem ról bądź ryzykiem utraty stanowiska w zespole. Właściwie poprowadzony proces wdrażania automatyzacji powinien wzbudzić bardzo duże zainteresowanie wszystkich członków zespołu i zachęcić ich do uczestniczenia w zmianach organizacyjno-technicznych.

Role i odpowiedzialności

Automatyzacja testów powinna być procesem, w którym może uczestniczyć każdy, chociaż nie oznacza to oczywiście, że wszyscy uczestnicy będą pełnić tę samą rolę. Zaprojektowanie, zaimplementowanie i utrzymanie zautomatyzowanego środowiska testowego jest zadaniem o charakterze technicznym, które powinno być realizowane wyłącznie przez osoby mające duże umiejętności programistyczne i dobre przygotowanie techniczne. W wyniku prac związanych z opracowywaniem testów automatycznych powinno powstać środowisko, z którego będą mogły korzystać w równym stopniu osoby posiadające umiejętności techniczne i ich nieposiadające. Do uzyskania maksymalnych korzyści w związku z wdrożeniem zautomatyzowanego środowiska testowego niezbędne są z kolei osoby mające wiedzę merytoryczną i umiejętności w zakresie testowania. Ich zadaniem będzie opracowanie odpowiednich skryptów testowych (wraz z odpowiednimi danymi testowymi), które posłużą do sterowania zautomatyzowanym środowiskiem i zapewnienia zakładanego pokrycia testowego. Kolejna grupa to eksperci merytoryczni (w zakresie danej domeny), którzy weryfikują raporty, aby potwierdzić prawidłowe funkcjonowanie aplikacji, oraz eksperci techniczni, którzy dbają o poprawne i efektywne działanie zautomatyzowanego środowiska. Ekspertami technicznymi mogą być również programiści zainteresowani testowaniem, ponieważ doświadczenie w dziedzinie wytwarzania oprogramowania jest niezbędne do zaprojektowania oprogramowania, które będzie łatwe w utrzymaniu, co jest niezmiernie istotne w kontekście automatyzacji testów. Programiści mogą skoncentrować się na strukturze do testów automatycznych (TAF) lub bibliotekach testowych, natomiast implementacja przypadków testowych powinna pozostać w gestii testerów.

Współpraca między programistami a inżynierami automatyzacji testów

Powodzenie automatyzacji testów zależy również od zaangażowania zespołu programistów i zespołu testerów. W przypadku automatyzacji konieczna jest dużo ściślejsza współpraca między programistami a testerami, w ramach której programiści udzielają testerom pomocy technicznej i informacji technicznych dotyczących stosowanych metod/narzędzi programistycznych, a inżynierowie automatyzacji testów mogą zgłaszać swoje obawy dotyczące testowalności projektów systemu i tworzonego kodu. Jest to szczególnie istotne w przypadku nieprzestrzegania standardów bądź stosowania przez programistów nietypowych, opracowanych we własnym zakresie lub zupełnie nowych bibliotek/obiektów (np. elementu sterującego GUI innej firmy, który może być niekompatybilny z wybranym narzędziem do automatyzacji). Na koniec należy zaznaczyć, że kierownicy projektu w organizacji muszą dysponować jednoznacznymi informacjami na temat ról i odpowiedzialności niezbędnych do pomyślnego wprowadzenia automatyzacji.

Równoległe wykonywanie prac

W ramach działań związanych ze zmianą sposobu testowania wiele organizacji powołuje pracujący równolegle zespół, który rozpoczyna automatyzowanie dotychczasowych skryptów testów manualnych. Po zautomatyzowaniu nowe skrypty są włączane do procesu testowania w miejsce skryptów wykonywanych manualnie. Zanim tak się jednak stanie, zaleca się porównanie skryptów testów automatycznych z zastępowanymi przez nie skryptami testów manualnych w celu sprawdzenia, czy wykonują one te same testy i walidacje.

W wielu przypadkach przed przekształceniem skryptów manualnych w automatyczne dokonuje się ich oceny. Może ona wykazać, że dotychczasowe manualne skrypty testowe wymagają restrukturyzacji, aby działały skuteczniej i bardziej efektywnie w warunkach automatyzacji.

Raportowanie na temat automatyzacji testów

Rozwiązanie dla testów automatycznych (TAS) może automatycznie generować różne raporty zawierające między innymi informacje o statusie poszczególnych skryptów lub kroków skryptów (zaliczony/niezaliczony), ogólne dane statystyczne dotyczące wykonywania testów oraz informacje o ogólnej wydajności TAS. Bardzo ważne jest również dostarczenie informacji na temat poprawności działania TAS, które pozwolą potwierdzić dokładność i kompletność wszelkich rezultatów dotyczących konkretnych aplikacji (patrz Rozdział 7. — Weryfikowanie rozwiązania dla testów automatycznych (TAS)).

6.2 Określenie kroków niezbędnych do implementacji zautomatyzowanych testów regresji

Testowanie regresji stwarza znakomitą okazję do zastosowania automatyzacji. Środowisko testowe regresji stale się poszerza, ponieważ dzisiejsze testy funkcjonalne stają się jutrzejszymi testami regresji. W tej sytuacji jest tylko kwestią czasu, kiedy liczba testów regresji przerośnie możliwości czasowe i kadrowe tradycyjnych zespołów testowych wykonujących testy manualne.

W ramach przygotowań do automatyzacji testów regresji należy zadać sobie kilka pytań:

- Jak często testy powinny być wykonywane?
- Ile trwa wykonywanie każdego testu i całego zestawu testów regresji?
- Czy w testach występują pokrywające się obszary funkcjonalne?
- Czy testy współużytkują dane?
- Czy istnieją wzajemne zależności między testami?
- Jakie warunki wstępne muszą zostać spełnione przed wykonaniem testu?
- Jakie jest procentowe pokrycie systemu podlegającego testowaniu przez testy?
- Czy testy są obecnie wykonywane poprawnie (bez awarii)?
- Co należy zrobić, jeśli testy regresji trwają zbyt długo?

Poniżej opisano dokładniej poszczególne kryteria.

Częstotliwość wykonywania testów

Do zautomatyzowania najlepiej nadają się testy, które są często wykonywane w ramach testowania regresji. Testy te zostały już opracowane i sprawdzają znaną funkcjonalność systemu podlegającego testowaniu, a zastosowanie automatyzacji pozwoli wydatnie skrócić czas ich wykonywania.

Czas wykonywania testów

Czas potrzebny do wykonania danego testu lub całego zestawu testowego jest ważnym parametrem, który należy wziąć pod uwagę przy ocenie korzyści wynikających z automatyzacji testów regresji. Implementowanie testów automatycznych można zacząć właśnie od czasochłonnych testów — umożliwi to szybsze i bardziej efektywne wykonywanie każdego testu, a także pozwoli dodać kolejne cykle automatycznych testów regresji. Wśród zalet takiego podejścia należy wymienić zwiększenie ilości informacji zwrotnych na temat jakości systemu podlegającego testowaniu i częstotliwości ich otrzymywania oraz zmniejszenie ryzyka związanego z wdrożeniem.

Pokrywające się obszary funkcjonalne

W przypadku automatyzowania istniejących testów regresji zaleca się zidentyfikowanie wszelkich pokrywających się obszarów funkcjonalnych (w poszczególnych przypadkach testowych i między różnymi przypadkami) oraz — jeśli jest to możliwe — ograniczenie ich w równoważnym teście automatycznym. Uzyskany w ten sposób wzrost efektywności w postaci skrócenia czasu wykonywania testów automatycznych będzie istotny w związku z wykonywaniem coraz większej liczby zautomatyzowanych przypadków testowych. Testy opracowywane z wykorzystaniem automatyzacji często przyjmują nową strukturę, ponieważ opierają się na komponentach wielokrotnego użytku i współużytkowanych repozytoriach danych. Dotychczasowe testy manualne są nierzadko rozkładane na kilka mniejszych testów automatycznych, a w innych przypadkach następuje konsolidacja kilku testów manualnych w jeden większy test automatyczny. Aby opracować skuteczną strategię przekształcania testów, należy dokonać oceny testów manualnych na poziomie indywidualnym i grupowym.

Współużytkowanie danych

Testy często współużytkują dane, czyli korzystają z tego samego rekordu danych do wykonywania różnych funkcji w systemie podlegającym testowaniu. Przykładem może być sytuacja, w której przypadek testowy „A” sprawdza liczbę niewykorzystanych dni urlopu pracownika, a przypadek testowy „B” sprawdza, jakie kursy pracownik ten odbył w ramach realizacji celów rozwoju zawodowego. Oba przypadki testowe odwołują się do danych tego samego pracownika, ale każdy weryfikuje inne parametry. W przypadku testowania manualnego dane pracownika zostałyby powielone w poszczególnych przypadkach testowych dla testów manualnych weryfikujących te dane, natomiast w przypadku testów automatycznych wspólne dane powinny być w miarę możliwości przechowywane i udostępniane w jednym miejscu, co pozwala uniknąć ich powielania i wprowadzania błędów.

Wzajemne zależności między testami

Podczas wykonywania złożonych scenariuszy testów regresji każdy test może być zależny od jednego lub kilku innych testów. Jest to dość powszechna sytuacja — przykładem może być wykorzystanie identyfikatora zamówienia utworzonego w wyniku wykonania jednego z kroków testu. Kolejne testy mogą sprawdzać, czy: (a) nowe zamówienie jest prawidłowo wyświetlane w systemie, (b) w zamówieniu można wprowadzać zmiany oraz (c) zamówienie można pomyślnie usunąć. W każdym przypadku konieczne jest zarejestrowanie wartości identyfikatora zamówienia (ang. *order ID*) utworzonej dynamicznie w pierwszym teście, aby można ją było wykorzystać ponownie w późniejszych testach. Sposób spełnienia tego wymogu zależy od projektu TAS.

Warunki wstępne związane z testami

Wykonanie testu jest często niemożliwe, dopóki nie zostaną spełnione określone warunki wstępne — takie jak wybór właściwej bazy danych lub właściwego zbioru danych testowych bądź ustawienie wartości/parametrów początkowych. Wiele kroków niezbędnych do zainicjowania środowiska i spełnienia warunków wstępnych testu można zautomatyzować, co pozwala uniknąć ryzyka ich pominięcia, a tym samym zwiększyć niezawodność i wiarygodność całego rozwiązania. W związku z automatyzowaniem testów regresji powyższe warunki wstępne należy również włączyć do ogólnego procesu automatyzacji.

Pokrycie systemu podlegającego testowaniu (SUT)

Za każdym razem, gdy są wykonywane testy, sprawdzają one część funkcjonalności systemu podlegającego testowaniu. Dlatego aby zbadać jakość całego systemu, należy zaprojektować testy w sposób zapewniający jak najszersze i jak najgłębsze pokrycie. Do monitorowania wykonywanych testów automatycznych i pomiaru ich skuteczności można użyć narzędzi mierzących pokrycie kodu. W przypadku automatycznego testowania regresji można oczekiwać, że dodatkowe testy zapewnią z czasem dodatkowe pokrycie (mierzenie pokrycia jest skutecznym sposobem na określenie wartości samych testów).

Poprawne wykonywanie testów

Przed przekształceniem manualnego testu regresji w test automatyczny należy sprawdzić, czy działa on prawidłowo. Dopiero wówczas można rozpocząć czynności mające na celu pomyślne przekształcenie danego testu manualnego w automatyczny test regresji. Jeśli test manualny nie jest wykonywany poprawnie (czy to dlatego, że został źle napisany, korzysta z niepoprawnych danych, jest nieaktualny lub nie został zsynchronizowany z aktualną wersją systemu podlegającego testowaniu, czy też na skutek defektu w SUT), należy najpierw rozpoznać i/lub usunąć podstawową przyczynę problemu. W przeciwnym razie po przekształceniu powstanie niedziałający test automatyczny, co oznacza stratę czasu i zasobów.

Duże zbiory testów regresji

Zbiór testów regresji związany z systemem podlegającym testowaniu może rozrosnąć się do sporych rozmiarów — na tyle dużych, że nie będzie go można wykonać w całości przez noc lub przez weekend.

W takiej sytuacji można rozważyć równoczesne wykonywanie przypadków testowych, o ile jest dostępnych kilka systemów podlegających testowaniu. W przypadku aplikacji na komputery PC nie stanowi to zwykle problemu, gorzej jest natomiast, jeśli systemem podlegającym testowaniu jest na przykład samolot lub rakieta kosmiczna. Słaba dostępność i/lub wysoki koszt SUT uniemożliwiają w praktyce równoczesne wykonywanie operacji. W takim przypadku rozwiązaniem może być wykonywanie testu regresji w częściach — z czasem (na przykład po kilku tygodniach) zostanie wówczas wykonany cały zbiór testów. Wyboru wykonywanej części zestawu testowego testów regresji można dokonać na podstawie analizy ryzyka (uwzględniającej na przykład to, które elementy systemu podlegającego testowaniu zostały ostatnio zmienione).

6.3 Czynniki, które należy wziąć pod uwagę przy automatyzowaniu testów nowych funkcjonalności (cech)

Przypadki testowe dotyczące nowej funkcjonalności są na ogół łatwiejsze do zautomatyzowania, gdy implementacja takiej funkcjonalności nie została jeszcze zakończona (a nawet lepiej — rozpoczęta). W tej sytuacji inżynier automatyzacji testów może skorzystać z posiadanej wiedzy, aby wyjaśnić programistom i architektom, co dokładnie należy uwzględnić w nowej funkcjonalności, aby można ją było skutecznie i efektywnie sprawdzać przy użyciu rozwiązania dla testów automatycznych (TAS).

W miarę wprowadzania w systemie podlegającym testowaniu nowych funkcji, testerzy muszą opracowywać nowe testy odpowiadające tym funkcjom i związanym z nimi wymaganiom. Aby ustalić, czy obecne rozwiązanie dla testów automatycznych będzie w stanie zaspokoić potrzeby związane z nową funkcjonalnością, inżynier automatyzacji testów (TAE) musi pozyskać informacje zwrotne od projektantów testów, którzy dysponują odpowiednią wiedzą merytoryczną. Przedmiotem analizy mogą być między innymi obecnie przyjęte podejście, stosowane narzędzia programistyczne innych firm, narzędzia testowe itd.

Zmiany w TAS należy analizować w kontekście dotychczasowych komponentów testaliów do testów automatycznych. Pozwoli to w pełni udokumentować wszystkie zmienione lub dodane funkcje oraz zadbać o to, aby nie wpływały one na zachowanie (lub wydajność) dotychczasowych funkcji rozwiązania.

Jeśli nowa funkcjonalność zostanie zaimplementowana, na przykład, przy użyciu innej klasy obiektu, może być konieczne zaktualizowanie lub uzupełnienie komponentów testaliów. Ponadto należy dokonać oceny kompatybilności z dotychczasowymi narzędziami testowymi i wskazać ewentualne rozwiązania zastępcze. W przypadku podejścia opartego na słowach kluczowych uwzględnienie nowej funkcjonalności może na przykład wymagać opracowania dodatkowych słów kluczowych lub zmodyfikowania/rozszerzenia istniejących słów kluczowych.

W pewnych sytuacjach może być również konieczne wypróbowanie dodatkowych narzędzi do testowania obsługujących nowe środowisko, w którym występuje nowa funkcjonalność. Nowe narzędzie może na przykład okazać się niezbędne, jeśli dotychczasowe narzędzie obsługuje jedynie język HTML.

Nowe wymagania związane z testowaniem mogą mieć wpływ na istniejące testy automatyczne i komponenty testaliów. Dlatego przed dokonaniem jakichkolwiek zmian należy uruchomić dotychczasowe testy automatyczne w odniesieniu do nowego/zaktualizowanego systemu podlegającego testowaniu, aby zweryfikować i zarejestrować ewentualne zmiany w działaniu testów (czynność ta powinna również objąć odwzorowanie wzajemnych zależności z innymi testami). Każda nowa zmiana technologiczna wiąże się z koniecznością przeanalizowania obecnych komponentów testaliów (w tym narzędzi testowych, bibliotek funkcji, interfejsów API itd.) oraz dotychczasowego TAS pod kątem kompatybilności.

Jeśli dotychczasowe wymagania ulegną zmianie, w harmonogramie projektu (strukturze podziału pracy) należy uwzględnić nakłady pracy związane z aktualizacją przypadków testowych służących do weryfikowania zgodności z tymi wymaganiami. W takim przypadku śledzenie powiązań między wymaganiami a odpowiednimi przypadkami testowymi pozwoli wskazać, które przypadki wymagają zaktualizowania (aktualizacje powinny być elementem całościowego planu).

Na koniec należy ustalić, czy dotychczasowe rozwiązanie dla testów automatycznych nadal zaspokaja potrzeby bieżącej wersji systemu podlegającego testowaniu. W tym celu należy zadać sobie pytanie, czy techniki implementacji są nadal właściwe, czy też wymagana jest nowa architektura, a jeśli tak, to czy można ją wdrożyć poprzez rozszerzenie obecnych funkcji.

Wprowadzanie nowej funkcjonalności jest dla inżynierów automatyzacji testów okazją do zadbania o testowalność nowo zdefiniowanych funkcji. Dlatego już w fazie projektowania należy wziąć pod uwagę potrzeby związane z testowaniem i zaplanować udostępnienie interfejsów testowych, za pośrednictwem których będzie można weryfikować nową funkcjonalność przy użyciu języków skryptowych lub narzędzia do automatyzacji testów. Więcej informacji na ten temat zawiera podrozdział 2.3 (Projektowanie pod kątem testowalności i niezawodności).

6.4 Czynniki, które należy wziąć pod uwagę przy implementacji automatycznych testów potwierdzających

Testowanie potwierdzające wykonuje się po wprowadzeniu w kodzie poprawki mającej na celu usunięcie zgłoszonego wcześniej defektu. Aby potwierdzić, że defekt został usunięty, tester wykonuje zwykle kroki niezbędne do jego odtworzenia.

Zdarza się, że defekty pojawiają się ponownie w kolejnych wersjach (co może sygnalizować problem z zarządzaniem konfiguracją), w związku z czym testy potwierdzające świetnie nadają się do zautomatyzowania. Automatyzacja pomaga skrócić czas wykonywania testów potwierdzających, przy czym testy te można dodać jako uzupełnienie do istniejącego środowiska testowego służącego do wykonywania automatycznych testów regresji.

Automatyczny test potwierdzający obejmuje zwykle wąski wycinek funkcjonalności. Test taki można zaimplementować w dowolnym momencie po zgłoszeniu defektu i określeniu kroków niezbędnych do jego odtworzenia. Automatyczne testy potwierdzające można włączyć do standardowego zestawu testów regresji lub, jeśli jest to uzasadnione, dodać do istniejących testów automatycznych. Oba podejścia pozwalają wykorzystać zalety automatycznych testów potwierdzających.

Śledzenie automatycznych testów potwierdzających umożliwia również raportowanie czasu i liczby cykli poświęconych na usuwanie defektów.

Aby upewnić się, że w związku z usunięciem danego defektu nie wprowadzono przypadkowo nowych defektów, oprócz testowania potwierdzającego należy również zastosować technikę testowania regresji (do określenia właściwego zakresu testowania regresji może być niezbędna analiza wpływu).

7. Weryfikowanie rozwiązania dla testów automatycznych (TAS) — 120 min.

Słowa kluczowe

weryfikacja

Cele nauczania — weryfikowanie rozwiązania dla testów automatycznych (TAS)

7.1 Weryfikowanie komponentów zautomatyzowanego środowiska testowego

ALTA-E-7.1.1 (K3) Kandydat potrafi sprawdzić poprawność zautomatyzowanego środowiska testowego, w tym konfiguracji narzędzi testowych.

7.2 Weryfikowanie zestawu testów automatycznych

ALTA-E-7.2.1 (K3) Kandydat potrafi sprawdzić poprawność zachowania danego skryptu testów automatycznych i/lub zestawu testów automatycznych.

7.1 Weryfikowanie komponentów zautomatyzowanego środowiska testowego

Zespół automatyzacji testów musi sprawdzić, czy zautomatyzowane środowisko testowe działa zgodnie z oczekiwaniami. Sprawdzenia dokonuje się np. przed rozpoczęciem testów automatycznych.

Aby sprawdzić prawidłowość działania komponentów zautomatyzowanego środowiska testowego, można wykonać kilka kroków, które szczegółowo opisano poniżej.

Instalowanie, konfigurowanie i dostosowywanie narzędzi testowych

Rozwiązanie dla testów automatycznych (TAS) składa się z wielu komponentów. Aby zapewnić niezawodność i powtarzalną wydajność całego rozwiązania, należy uwzględnić każdy z nich. Sercem TAS są komponenty wykonywalne wraz z odpowiadającymi im bibliotekami funkcji oraz dane pomocnicze i pliki konfiguracyjne. Proces konfigurowania TAS może przebiegać w różny sposób: od użycia automatycznych skryptów instalacyjnych, po manualne umieszczenie plików w odpowiednich folderach. Należy przy tym pamiętać, że w przypadku narzędzi do testowania — podobnie jak w przypadku systemów operacyjnych i innych aplikacji — regularnie udostępniane są pakiety serwisowe, a do zapewnienia kompatybilności z danym środowiskiem systemu podlegającemu testowaniu mogą być niezbędne opcjonalne lub obowiązkowe dodatki.

Automatyczne instalowanie (lub kopiowanie) z centralnego repozytorium ma określone zalety, takie jak możliwość zagwarantowania, że testy w różnych systemach podlegających testowaniu będą wykonywane przy użyciu tej samej wersji oraz (stosownie do okoliczności) tej samej konfiguracji TAS. Kolejną zaletą jest możliwość aktualizowania TAS za pośrednictwem repozytorium. Sposób korzystania z repozytorium i proces aktualizacji TAS do nowej wersji powinny być takie same, jak w przypadku standardowych narzędzi programistycznych.

Skrypty testowe zawierające typowo zaliczane/niezaliczane testy

Niezaliczenie przypadków testowych, które zwykle kończą się powodzeniem, jednoznacznie sygnalizuje wystąpienie poważnego problemu, który należy jak najszybciej rozwiązać. Z kolei zaliczenie przypadków testowych, które powinny zakończyć się wynikiem negatywnym, oznacza konieczność zidentyfikowania komponentu, który nie zadziałał prawidłowo. W obu przypadkach ważne jest sprawdzenie poprawności generowania plików dzienników i metryk wydajności oraz procesu automatycznego ustanawiania i rozmontowywania przypadków/skryptów testowych. Ponadto zalecane jest wykonanie kilku testów reprezentujących różne typy i poziomy testów (testy funkcjonalne, testy wydajnościowe, testy komponentów itd.), przy czym testy te należy również wykonać na poziomie struktury.

Powtarzalność ustanawiania i rozmontowywania środowiska testowego

Rozwiązanie dla testów automatycznych będzie implementowane w różnych systemach i na różnych serwerach. Aby zagwarantować jego poprawne działanie w poszczególnych środowiskach, należy podejść w usystematyzowany sposób do kwestii wdrażania i usuwania TAS w danym środowisku. Gdy proces ten przebiega prawidłowo, budowa i przebudowa rozwiązania dla testów automatycznych nie powoduje zauważalnych różnic w sposobie jego działania w obrębie poszczególnych środowisk i między wieloma środowiskami. Aby zagwarantować niezawodne tworzenie danej konfiguracji, należy zastosować odpowiednie mechanizmy zarządzania konfiguracją komponentów TAS.

Konfiguracja środowiska testowego i jego komponentów

Zrozumienie i udokumentowanie poszczególnych komponentów tworzących TAS pozwala uzyskać niezbędną wiedzę o tym, na które obszary tego rozwiązania mogą wpływać zmiany w środowisku systemu podlegającego testowaniu lub które z tych obszarów mogą wymagać modyfikacji na skutek takich zmian.

Połączenia z systemami/interfejsami wewnętrznymi i zewnętrznymi

Po zainstalowaniu TAS w danym środowisku systemu podlegającego testowaniu, ale jeszcze przed rozpoczęciem faktycznych testów, należy wykonać szereg czynności sprawdzających spełnienie warunków wstępnych dotyczących dostępności połączeń z wewnętrznymi i zewnętrznymi systemami, interfejsami itd. Określenie warunków wstępnych dotyczących automatyzacji jest ważnym elementem mającym na celu zapewnienie poprawności instalacji i konfiguracji TAS.

Inwazyjność narzędzi do testów automatycznych

Rozwiązanie dla testów automatycznych jest często ściśle sprzężone z systemem podlegającym testowaniu. Ma to z założenia zapewnić wysoki poziom kompatybilności, zwłaszcza w przypadku interakcji przy użyciu interfejsu graficznego (GUI). Ścisła integracja może mieć jednak również negatywne skutki, takie jak zmiana zachowania systemu podlegającego testowaniu, gdy w jego środowisku znajduje się TAS, różnice w zachowaniu systemu podlegającego testowaniu przy testowaniu automatycznym i przy obsłudze manualnej bądź spadek wydajności SUT, gdy TAS znajduje się w jego środowisku lub wykonuje w nim testy.

Poziom inwazyjności zależy od wybranego podejścia do automatyzacji testów. Na przykład:

- W przypadku komunikowania się z systemem podlegającym testowaniu na poziomie interfejsów zewnętrznych poziom inwazyjności jest bardzo niski. Interfejsy zewnętrzne mogą korzystać z sygnałów elektronicznych (generowanych np. przez fizyczne przełączniki) lub transmisji USB (w przypadku urządzeń USB, np. klawiatur).³ To podejście pozwala najlepiej symulować czynności wykonywane przez docelowego użytkownika, ponieważ oprogramowanie systemu podlegającego testowaniu nie jest w żaden sposób modyfikowane do celów testowania, a testy nie wpływają na zachowanie systemu i parametry czasowe jego pracy. Z drugiej strony, komunikowanie się z systemem podlegającym testowaniu w ten sposób bywa bardzo skomplikowane i może wymagać wyspecjalizowanego sprzętu, obsługi określonych języków opisu sprzętu itd. Podejście to nie jest typowe w przypadku systemów złożonych wyłącznie z oprogramowania, bywa natomiast stosowane w przypadku produktów z oprogramowaniem wbudowanym.
- W przypadku komunikowania się z systemem podlegającym testowaniu na poziomie interfejsu graficznego (GUI) środowisko tego systemu jest dostosowywane, aby umożliwić wstrzykiwanie poleceń interfejsu użytkownika i wyodrębnianie informacji wymaganych przez przypadki testowe. Testy nie zmieniają bezpośrednio zachowania systemu podlegającego testowaniu, ale wpływają na parametry czasowe, co może mieć pośredni wpływ na jego działanie. Poziom inwazyjności jest wyższy niż w poprzednim punkcie, ale komunikowanie się z systemem podlegającym testowaniu jest mniej skomplikowane. W przypadku tego rodzaju automatyzacji można często skorzystać z oprogramowania do powszechnej sprzedaży.
- Komunikacja z systemem podlegającym testowaniu może odbywać się za pośrednictwem interfejsów testowych w oprogramowaniu lub przy użyciu dotychczasowych interfejsów oferowanych przez to oprogramowanie. Dostępność takich interfejsów (tj. API) jest istotnym elementem projektowania pod kątem testowalności. W tym przypadku poziom inwazyjności może być dość wysoki, ponieważ testy automatyczne wykorzystują interfejsy, z których mogą wcale nie korzystać użytkownicy systemu (interfejsy testowe), bądź interfejsy, które w normalnych warunkach mogą być używane w innym kontekście. Z drugiej strony, wykonywanie testów automatycznych za pośrednictwem interfejsów API jest bardzo łatwe i tanie. Korzystanie z interfejsów testowych w trakcie testowania SUT może być uzasadnione pod warunkiem znajomości potencjalnego ryzyka.

Testowanie z wysokim poziomem inwazyjności pozwala ujawnić awarie, które nie są widoczne w warunkach codziennej eksploatacji. Jeśli jednak rezultatem tego będą niezaliczone testy automatyczne, zaufanie do TAS może ulec znacznemu obniżeniu. W tej sytuacji programiści mogą zażądać manualnego odtworzenia awarii zidentyfikowanych w ramach testowania automatycznego (o ile jest to możliwe) na potrzeby analizy.

Testowanie komponentów struktury (*framework*)

Podobnie jak w przypadku innych projektów związanych z wytwarzaniem oprogramowania, elementy struktury do automatyzacji (ang. *framework*) powinny być indywidualnie przetestowane i zweryfikowane. Proces ten może obejmować zarówno testowanie funkcjonalne, jak i нефункционалне (tj. testowanie wydajnościowe, testowanie zużycia zasobów, testowanie użyteczności itd.).

Na przykład, komponenty odpowiedzialne za weryfikację obiektów w systemach z interfejsem graficznym (GUI) muszą zostać przetestowane pod kątem obsługi szerokiej gamy klas obiektów w celu potwierdzenia, że weryfikacja obiektów przebiega prawidłowo. Podobnie jest w przypadku dzienników błędów i raportów — w tym przypadku należy sprawdzić, czy dostarczają one dokładnych informacji na temat statusu automatyzacji i zachowania systemu podlegającego testowaniu.

³ Wyjaśnienie autora, Armina Borna: gdy tworzone są w SUT interfejsy dedykowane do automatyzacji testów (a nie po to, by wspierać inne wymagania), wtedy poziom inwazyjności jest wysoki. Gdy interfejsy wbudowane w SUT używane są po to, żeby wspierać dowolne wymagania (także automatyzację testów), wtedy poziom inwazyjności jest niski. Podczas korzystania z SUT z już istniejących zewnętrznych systemów, poziom inwazyjności będzie bardzo niski. Zewnętrzne interfejsy mogą być elektronicznymi sygnałami (dla fizycznych przełączników), sygnałami USB dla urządzeń USB (takich jak klawiatura).

Testy niiefunkcjonalne pozwalają, między innymi, zrozumieć przyczyny spadku wydajności struktury oraz wykryć nieprawidłowości w zakresie wykorzystania zasobów systemowych, które mogą być objawem problemów takich jak wyciek pamięci. Ponadto sprawdzają współdziałanie komponentów w ramach struktury i/lub poza nią.

7.2 Weryfikowanie zestawu testów automatycznych

Zestawy testów automatycznych należy przetestować pod kątem kompletności, spójności i poprawnego działania. Aby sprawdzić, czy zestaw testów automatycznych działa w danym momencie prawidłowo, lub aby ustalić, czy jest zdalny do użytku, można zastosować różne procedury weryfikacji.

W celu zweryfikowania prawidłowości działania zestawu testów automatycznych można wykonać kilka kroków takich jak:

- wykonanie skryptów testowych ze znanym wynikiem (zaliczone/niezaliczone),
- sprawdzenie zestawu testowego,
- zweryfikowanie nowych testów, które koncentrują się na nowych funkcjach struktury,
- uwzględnienie powtarzalności testów,
- sprawdzenie, czy zestaw testów automatycznych zawiera wystarczającą liczbę punktów weryfikacji.

Poniżej opisano dokładniej poszczególne kryteria.

Wykonanie skryptów testowych ze znanym wynikiem (zaliczone/niezaliczone)

Niezaliczenie przypadków testowych, które zwykle kończą się powodzeniem, jednoznacznie sygnalizuje wystąpienie poważnego problemu, który należy jak najszybciej rozwiązać. Z kolei zaliczenie zestawu testowego, który powinien zakończyć się wynikiem negatywnym, oznacza konieczność zidentyfikowania przypadku testowego, który nie zadziałał prawidłowo. W obu przypadkach ważne jest sprawdzenie poprawności generowania plików dzienników i danych dotyczących wydajności oraz procesu automatycznego ustanawiania i rozmontowywania przypadków/skryptów testowych. Ponadto, zalecane jest wykonanie kilku testów reprezentujących różne typy i poziomy testów (testy funkcjonalne, testy wydajnościowe, testy komponentów itd.).

Sprawdzenie zestawu testowego

Należy sprawdzić, czy zestaw testowy jest kompletny (tj. czy określono oczekiwane rezultaty w odniesieniu do wszystkich przypadków testowych, czy są dostępne dane testowe itd.) oraz czy jego wersja odpowiada wersji struktury i systemu podlegającego testowaniu.

Zweryfikowanie nowych testów, które koncentrują się na nowych funkcjach struktury

Jeśli nowa funkcja TAS jest używana po raz pierwszy w przypadkach testowych, należy ją zweryfikować i ściśle monitorować pod kątem prawidłowego działania.

Uwzględnienie powtarzalności testów

W przypadku powtarzania testów rezultat/werdykt powinien być zawsze ten sam. Jeśli w zbiorze testów znajdują się przypadki testowe, które nie generują wiarygodnych rezultatów (np. w sytuacji wyścigu), przypadki te można przenieść poza aktywny zestaw testów automatycznych i oddzielnie przeanalizować w celu wykrycia podstawowej przyczyny problemu. W przeciwnym razie konieczne będzie poświęcenie dodatkowego czasu na wielokrotne wykonywanie tych samych przebiegów testów w celu przeanalizowania problemu.

Sporadyczne awarie również wymagają zbadania. Przyczyna problemu może leżeć po stronie przypadku testowego lub po stronie struktury, a nawet po stronie systemu podlegającego testowaniu. W takim przypadku w zidentyfikowaniu podstawowej przyczyny problemu może pomóc analiza plików dzienników (dotyczących przypadku testowego, struktury i systemu podlegającego testowaniu), ale może być również konieczne debugowanie. Ponadto przy poszukiwaniu podstawowej przyczyny, może być niezbędne skorzystanie z pomocy analityka testów, programisty lub eksperta merytorycznego.

Sprawdzenie, czy zestaw testów automatycznych i/lub przypadków testowych zawiera wystarczającą liczbę punktów weryfikacji

Musi istnieć możliwość sprawdzenia, czy zestaw testów automatycznych został wykonany, a jego wykonanie przyniosło oczekiwane rezultaty. Ponadto niezbędne są dowody potwierdzające, że zestaw testów i/lub przypadki testowe zostały wykonane zgodnie z oczekiwaniami. W celu uzyskania takich dowodów można między innymi zarejestrować dane na początku i na końcu każdego przypadku testowego, udokumentować status wykonania testu w przypadku każdego ukończonego przypadku testowego, zweryfikować osiągnięcie warunków wyjściowych itd.

8. Ciągłe doskonalenie — 150 min.

Słowa kluczowe

pielęgnacja

Cele nauczania — ciągłe doskonalenie

8.1 Możliwości wprowadzania udoskonaleń w zakresie automatyzacji testów

ALTA-E-8.1.1 (K4) Kandydat potrafi przeanalizować aspekty techniczne wdrożonego rozwiązania dla testów automatycznych i przedstawić zalecenia dotyczące udoskonaleń.

8.2 Dostosowywanie mechanizmów automatyzacji testów do zmian w środowisku i systemie podlegającym testowaniu

ALTA-E-8.2.1 (K4) Kandydat potrafi przeanalizować testalia do testów automatycznych (w tym komponenty środowiska testowego, narzędzia i wspomagające je biblioteki funkcji) w celu wskazania obszarów wymagających konsolidacji lub aktualizacji po wprowadzeniu określonego zbioru zmian w środowisku testowym lub w systemie podlegającym testowaniu.

8.1 Możliwości wprowadzania udoskonaleń w zakresie automatyzacji testów

Oprócz bieżących prac związanych z pielęgnacją, które są niezbędne do zapewnienia ciągłej synchronizacji rozwiązania dla testów automatycznych (TAS) z systemem podlegającym testowaniu, istnieje zwykle wiele innych okazji do udoskonalania tego rozwiązania. Wprowadzanie udoskonaleń w TAS może służyć osiągnięciu różnych korzyści, takich jak zwiększenie efektywności (a tym bardziej dalsze ograniczenie ręcznych interwencji), ułatwienie obsługi, wprowadzenie dodatkowych funkcji bądź usprawnienie obsługi czynności testowych. Przy podejmowaniu decyzji o sposobie udoskonalania TAS należy kierować się tym, jakie zmiany przyniosą największe korzyści z punktu widzenia projektu.

Udoskonalenia można wprowadzać w takich obszarach jak: tworzenie skryptów, weryfikacja, architektura, przetwarzanie początkowe i końcowe, dokumentacja czy obsługa narzędzi. Poniżej opisano dokładniej poszczególne obszary.

Tworzenie skryptów

Podejścia do tworzenia skryptów mogą być bardzo różne — od prostego podejścia ustrukturalizowanego poprzez podejścia sterowane danymi po bardziej zaawansowane podejścia oparte na słowach kluczowych (patrz sekcja 3.2.2). W niektórych przypadkach może być uzasadnione zaktualizowanie obecnego podejścia do tworzenia skryptów w TAS w odniesieniu do wszystkich nowych testów automatycznych. Nowe podejście można też zastosować wstecznie do wszystkich istniejących dotychczasowych testów automatycznych, a przynajmniej do tych, których utrzymanie jest najbardziej czasochłonne.

Zamiast zmieniać całe podejście do tworzenia skryptów, w ramach udoskonalania TAS można skoncentrować się na sposobie ich implementacji. Na przykład:

- Oszacowanie pokrywających się obszarów przypadków/kroków/procedur testowych w celu skonsolidowania testów automatycznych.
Przypadki testowe zawierające podobne sekwencje akcji nie powinny wielokrotnie implementować tych samych kroków. Kroki te należy przekształcić w funkcje i dodać do biblioteki, co następnie umożliwi ich wielokrotne wykorzystanie. Funkcje dostępne w bibliotece mogą być używane w ramach różnych przypadków testowych, co zwiększa utrzymywalność testaliów. Jeśli kroki są podobne, ale nie identyczne, może być konieczna parametryzacja.
Podejście to jest typowe dla testowania opartego na słowach kluczowych.
- Ustanowienie procesu odzyskiwania sprawności po wystąpieniu błędów w TAS i systemie podlegającym testowaniu.
Jeśli podczas wykonywania przypadków testowych wystąpi błąd, rozwiązanie dla testów automatycznych powinno odzyskać sprawność i przejść do następnego przypadku testowego. Jeśli błąd wystąpi w systemie podlegającym testowaniu, TAS musi wykonać w nim niezbędne czynności mające na celu przywrócenie sprawności (takie jak np. ponowne uruchomienie całego systemu podlegającego testowaniu).
- Przeanalizowanie mechanizmów oczekiwania w celu upewnienia się, że stosowany jest najlepszy ich typ. Poniżej opisano trzy często używane mechanizmy oczekiwania:
 1. Oczekiwanie przez ustalony czas (tj. określoną liczbę milisekund) może być podstawową przyczyną wielu problemów związanych z automatyzacją testów.
 2. Oczekiwanie dynamiczne z sondowaniem (tj. sprawdzaniem, czy nastąpiła określona zmiana stanu lub akcja) jest znacznie bardziej elastyczne i efektywne, ponieważ:
 - oczekiwanie trwa tylko tak długo, jak długo jest to konieczne, dzięki czemu nie marnuje się czasu testowania,
 - jeśli z jakiegoś powodu proces trwa dłużej, funkcja sondowania czeka na spełnienie warunku (należy również pamiętać o uwzględnieniu mechanizmu limitu czasu, ponieważ w przeciwnym razie po wystąpieniu problemu test mógłby trwać w nieskończoność).
 3. Jeszcze lepszym rozwiązaniem jest subskrypcja mechanizmu zdarzeń generowanych przez system podlegający testowaniu. Opcja ta jest znacznie bardziej niezawodna niż pozostałe dwie, jednak w takim przypadku język skryptowy używany do wykonywania testów musi obsługiwać subskrypcję zdarzeń, a system podlegający testowaniu musi udostępniać te zdarzenia aplikacji testowej. Należy również pamiętać o uwzględnieniu limitu czasu, ponieważ w przeciwnym razie po wystąpieniu problemu test mógłby trwać w nieskończoność.
- Traktowanie testaliów jako oprogramowania.
Wytwarzanie i utrzymanie testaliów jest jedną z form wytwarzania oprogramowania, w związku z czym wymaga stosowania dobrych praktyk w dziedzinie tworzenia kodu (np. stosowania wytycznych

dotyczących tworzenia kodu oraz przeprowadzania analizy statycznej i przeglądów kodu). Może być nawet wskazane skorzystanie z usług programistów (a nie inżynierów automatyzacji testów) przy tworzeniu niektórych elementów testaliów (np. bibliotek).

- Ocena dotychczasowych skryptów pod kątem skorygowania lub wyeliminowania.
Zaleca się przeprojektowanie skryptów, które stwarzają problemy (takie jak sporadyczne błędy czy wysokie koszty utrzymania), a także usunięcie z zestawu testów skryptów, które nie przynoszą już żadnych korzyści.

Wykonywanie testów

To, że wykonywanie zestawu automatycznych testów regresji może nie zakończyć się np. w ciągu jednej nocy, nie powinno być zaskoczeniem. Jeśli jednak testowanie trwa zbyt długo, może być wskazane wykonanie testów równocześnie w kilku systemach, chociaż nie zawsze jest to możliwe. W przypadku drogich systemów może się okazać, że całość testowania musi zostać przeprowadzona tylko na jednej instalacji systemu. W rezultacie, może być konieczne podzielenie zestawu testów regresji na kilka części, z których każda będzie wykonywana w określonym czasie (np. przez jedną noc). Dalsza analiza pokrycia testów automatycznych może też ujawnić duplikację, której wyeliminowanie pozwoli skrócić czas wykonywania testów i zwiększyć efektywność w innych obszarach.

Weryfikacja

Przed utworzeniem nowych funkcji weryfikacji należy przyjąć zbiór standardowych metod weryfikacji, które mają być używane w ramach wszystkich testów automatycznych. Pozwoli to uniknąć wielokrotnego implementowania akcji związanych z weryfikacją w różnych testach. Jeśli metody weryfikacji są podobne, ale nie identyczne, zastosowanie parametryzacji pozwoli stworzyć funkcję przeznaczoną do wykorzystania przez wiele typów obiektów.

Architektura

W związku z wprowadzaniem udoskonaleń zwiększających testowalność systemu podlegającego testowaniu, może być konieczne dokonanie zmian w architekturze tego systemu i/lub architekturze automatyzacji. Działania takie mogą przynieść znaczne usprawnienie automatyzacji testów, ale modyfikacje po stronie systemu podlegającego testowaniu i TAS mogą wiązać się z dużymi inwestycjami. Na przykład, zmiana w systemie podlegającym testowaniu mająca na celu udostępnienie interfejsów API do celów testowania wymaga również wprowadzenia odpowiednich zmian w TAS. Dodawanie tego rodzaju funkcji na późniejszym etapie może być dość kosztowne, dlatego znacznie lepszym rozwiązaniem jest uwzględnienie ich na początku procesu automatyzacji (i na wczesnych etapach rozwoju systemu podlegającego testowaniu, patrz podrozdział 2.3 — Projektowanie pod kątem testowalności i automatyzacji).

Przetwarzanie wstępne i końcowe

Należy określić standardowe zadania związane z procesami ustanawiania i rozmontowania (ang. *setup and teardown*), zwane także przetwarzaniem wstępnym (ustanawianie – ang. *setup*) i końcowym (rozmontowanie – ang. *teardown*). Dzięki temu można uniknąć wielokrotnego implementowania tych samych zadań w ramach poszczególnych testów automatycznych, a tym samym obniżyć koszty utrzymania oraz zmniejszyć nakład pracy potrzebny do implementacji nowych testów automatycznych.

Dokumentacja

Ta kategoria obejmuje wszelkiego rodzaju dokumentację: od dokumentacji skryptów (wyjaśniającej ich działanie, sposób używania itd.) poprzez dokumentację dla użytkowników TAS po raporty i dzienniki tworzone przez to rozwiązanie.

Cechy TAS

Do TAS mogą być dodawane kolejne cechy i funkcje umożliwiające szczegółowe raportowanie, prowadzenie dzienników, integrację z innymi systemami itd. Funkcje takie należy dodawać wyłącznie pod warunkiem, że będą faktycznie używane, ponieważ dodawanie zbędnych funkcji zwiększa tylko złożoność rozwiązania, a w rezultacie zmniejsza jego niezawodność i utrzymywalność.

Aktualizacje i uaktualnienia TAS

W ramach aktualizacji TAS lub uaktualnienia TAS do nowej wersji mogą zostać udostępnione nowe funkcje przeznaczone do wykorzystania przez przypadki testowe (a także mogą zostać usunięte ewentualne przyczyny awarii). Wiąże się to jednak z pewnym ryzykiem, ponieważ aktualizacja struktury — polegająca na uaktualnieniu dotychczasowych narzędzi testowych lub wprowadzeniu nowych — może negatywnie wpłynąć na istniejące przypadki testowe. Dlatego przed wdrożeniem nowej wersji narzędzia testowego należy

wykonać za jego pomocą kilka przykładowych testów, które powinny być reprezentatywne dla różnych aplikacji objętych testowaniem automatycznym, różnych typów testów oraz (jeśli wymaga tego sytuacja) różnych środowisk.

8.2 Planowanie implementacji udoskonaleń w zakresie automatyzacji testów

Zmiany w istniejącym rozwiązaniu dla testów automatycznych (TAS) wymagają starannego zaplanowania i przeanalizowania. Stworzenie stabilnego TAS — złożonego ze struktury do testów automatycznych (TAF) oraz bibliotek komponentów — pochłonęło wiele wysiłku, a każda zmiana, choćby najbardziej trywialna, może mieć poważny wpływ na jego niezawodność i wydajność.

Identyfikowanie zmian w komponentach środowiska testowego

Należy ocenić, jakie zmiany i udoskonalenia powinny zostać wprowadzone, a następnie zastanowić się, czy wymagają one zmodyfikowania oprogramowania testującego, niestandardowych bibliotek funkcji, systemu operacyjnego itd., ponieważ każda taka zmiana ma wpływ na sposób działania TAS. Najważniejsze jest to, aby testy automatyczne były nadal wykonywane w sposób efektywny. W związku z powyższym, zmiany należy wprowadzać przyrostowo, co umożliwi mierzenie ich wpływu na TAS z wykorzystaniem ograniczonego zbioru skryptów testowych. Po stwierdzeniu, że nie występują żadne niekorzystne skutki, można przejść do pełnej implementacji zmian. Ostatnim krokiem mającym na celu sprawdzenie, czy zmiana nie wpłynęła negatywnie na skrypty automatyczne, jest wykonanie pełnego przebiegu testowania regresji. Jeśli podczas wykonywania skryptów regresji zostaną wykryte błędy, należy zidentyfikować ich podstawową przyczynę (na podstawie raportów, dzienników, analizy danych itd.), aby upewnić się, że nie są one skutkiem działań związanych z usprawnianiem automatyzacji.

Zwiększanie efektywności i skuteczności podstawowych bibliotek funkcji TAS

W miarę dojrzewania TAS odkrywane są nowe sposoby bardziej efektywnego wykonywania zadań. Powyższe nowe techniki (związane z optymalizacją kodu funkcji, zastosowaniem nowszych bibliotek systemu operacyjnego itd.) należy włączyć do podstawowych bibliotek funkcji wykorzystywanych w bieżącym projekcie i wszystkich innych projektach.

Ukierunkowanie na konsolidację wielu funkcji wykonujących operacje przy użyciu elementów sterujących tego samego typu

Dużą część działań wykonywanych podczas przebiegu testu automatycznego stanowi odpytywanie elementów sterujących w interfejsie graficznym (GUI) w celu uzyskania informacji na temat danego elementu (np. widoczny/niewidoczny, włączony/niewłączony, wielkość/wymiary, dane itd.). Na podstawie tych informacji test automatyczny może wybrać element z listy rozwijanej, wprowadzić dane w polu, odczytać wartość z pola itd. Istnieje kilka rodzajów funkcji, które mogą oddziaływać na elementy sterujące w celu uzyskania powyższych informacji. Niektóre z nich są bardzo ściśle wyspecjalizowane, inne zaś mają charakter bardziej ogólny. Oprócz wyspecjalizowanej funkcji obsługującej tylko listy rozwijane może być również dostępna (lub może zostać utworzona do użytku w TAS) funkcja współpracująca z kilkoma innymi funkcjami wskazywanymi przy użyciu parametru. Może się zatem okazać, że inżynier automatyzacji testów (TAE) korzysta z kilku funkcji, które dałoby się skonsolidować w ramach mniejszej liczby funkcji, uzyskując te same rezultaty, a jednocześnie zmniejszając wymagania związane z pielęgnacją.

Przebudowa architektury automatyzacji testów (TAA) w celu uwzględnienia zmian w systemie podlegającym testowaniu

Przez cały okres eksploatacji TAS konieczne jest wprowadzanie modyfikacji odzwierciedlających zmiany w systemie podlegającym testowaniu. W miarę rozwoju i dojrzewania systemu podlegającego testowaniu bazowa architektura testów automatycznych (TAA) musi również się rozwijać, aby zapewnić funkcje niezbędne do jego obsługi. Przy rozszerzaniu funkcjonalności należy zachować ostrożność, aby nowe funkcje nie były implementowane w sposób prowizoryczny: każda funkcja powinna zostać przeanalizowana, a zmiany powinny być wprowadzane na poziomie architektury zautomatyzowanego rozwiązania. Takie podejście gwarantuje, że w momencie wprowadzenia nowych funkcji systemu podlegającego testowaniu, które wymagają dodatkowych skryptów, będą już dostępne odpowiednie komponenty umożliwiające dodanie nowych testów automatycznych.

Konwencje nazewnictwa i standaryzacja

W miarę wprowadzania kolejnych zmian należy dbać o spójność konwencji nazewnictwa nowego kodu automatyzacji i nowych bibliotek funkcji z wcześniej zdefiniowanymi standardami (patrz sekcja 4.3.2 — Zakres i podejście).

Ocena istniejących skryptów dotyczących systemu podlegającego testowaniu pod kątem zmiany/eliminacji

Proces wprowadzania zmian i udoskonaleń obejmuje również dokonywanie oceny istniejących skryptów pod kątem wykorzystania i dalszej przydatności. W przypadku skomplikowanych i czasochłonnych testów bardziej opłacalne i efektywne może być na przykład dekompozycja ich na kilka mniejszych testów. Ponadto warto zastanowić się nad wyeliminowaniem testów, które są wykonywane rzadko lub nie są wcale stosowane, ponieważ pozwala to zmniejszyć złożoność TAS oraz uzyskać bardziej przejrzysty obraz wymagań związanych z utrzymaniem.

9. Bibliografia

9.1 Standardy

Wśród standardów dotyczących automatyzacji testów należy wymienić w szczególności:

- notację Testing and Test Control Notation (TTCN-3) opracowaną przez Europejski Instytut Norm Telekomunikacyjnych (ETSI) i Międzynarodowy Związek Telekomunikacyjny (ITU), w skład której wchodzi następujące dokumenty:
 - ES 201 873-1: TTCN-3 Core Language
 - ES 201 873-2: TTCN-3 Tabular Presentation Format (TFT)
 - ES 201 873-3: TTCN-3 Graphical Presentation Format (GFT)
 - ES 201 873-4: TTCN-3 Operational Semantics
 - ES 201 873-5: TTCN-3 Runtime Interface (TRI)
 - ES 201 873-6: TTCN-3 Control Interface (TCI)
 - ES 201 873-7: Using ASN.1 with TTCN-3
 - ES 201 873-8: Using IDL with TTCN-3
 - ES 201 873-9: Using XML with TTCN-3
 - ES 202 781: Extensions: Configuration and Deployment Support
 - ES 202 782: Extensions: TTCN-3 Performance and Real-Time Testing
 - ES 202 784: Extensions: Advanced Parameterization
 - ES 202 785: Extensions: Behaviour Types
 - ES 202 786: Extensions: Support of interfaces with continuous signals
 - ES 202 789: Extensions: Extended TRI
- specyfikację Automatic Test Markup Language (ATML) opracowaną przez IEEE (Institute of Electrical and Electronics Engineers), która określa wymagania w zakresie
 - IEEE Std 1671.1: Test Description
 - IEEE Std 1671.2: Instrument Description
 - IEEE Std 1671.3: UUT Description
 - IEEE Std 1671.4: Test Configuration Description
 - IEEE Std 1671.5: Test Adaptor Description
 - IEEE Std 1671.6: Test Station Description
 - IEEE Std 1641: Signal and Test Definition
 - IEEE Std 1636.1: Test Results
- ISO/IEC/IEEE 29119-3
- UML Testing Profile (UTP) by OMG (Object Management Group) określające koncepcje specyfikacji testów dla:
 - architektury testów,
 - danych testowych,
 - zachowania testów,
 - rejestrowania testów,
 - zarządzania testami.

9.2 Dokumenty ISTQB®

<u>Identyfikator</u>	<u>Przywoływany dokument</u>
ISTQB-AL-TM	ISTQB® Certyfikowany tester. Sylabus poziomu zaawansowanego. Kierownik Testów, wersja 2012 (dokument dostępny w serwisie [ISTQB-Web])
ISTQB-AL-TTA	ISTQB® Certyfikowany tester. Sylabus poziomu zaawansowanego. Techniczny Analitik Testów, wersja 4.0, 2021 (dokument dostępny w serwisie [ISTQB-Web])
ISTQB-EL-CEP	ISTQB® Rozszerzenie certyfikacji poziomu zaawansowanego (dokument dostępny w serwisie [ISTQB-Web])
ISTQB-EL-Modules	ISTQB® Przegląd modułów poziomu zaawansowanego, wersja 1.2 z 23 sierpnia 2013 r. (dokument dostępny w serwisie [ISTQB-Web])
ISTQB-EL-TM	ISTQB® Sylabus poziomu zaawansowanego. Zarządzanie testami, wersja 2011 (dokument dostępny w serwisie [ISTQB-Web])
ISTQB-FL	ISTQB® Sylabus poziomu podstawowego, wersja 2018, v. 3.1 (dokument dostępny w serwisie [ISTQB-Web])
ISTQB-Glossary	ISTQB® Słownik terminów testowych ISTQB®, aplikacja dostępna pod adresem: https://glossary.istqb.org

9.3 Znaki towarowe

W niniejszym dokumencie używane są następujące zastrzeżone znaki towarowe i znaki usług: nazwa ISTQB® jest zastrzeżonym znakiem towarowym International Software Testing Qualifications Board.

9.4 Książki

<u>Identyfikator</u>	<u>Przywoływana książka</u>
[Baker08]	Paul Baker, Zhen Ru Dai, Jens Grabowski, Ina Schieferdecker, „Model-Driven Testing: Using the UML Testing Profile”, Springer 2008, ISBN-10: 3540725628, ISBN-13: 978-3540725626
[Dustin09]	Efiede Dustin, Thom Garrett i Bernie Gauf, „Implementing Automated Software Testing: how to save time and lower costs while raising quality”, Addison-Wesley 2009, ISBN 0-321-58051-6
[Dustin99]	Efiede Dustin, Jeff Rashka, John Paul, „Automated Software Testing: introduction, management, and performance”, Addison-Wesley 1999, ISBN-10: 0201432870, ISBN-13: 9780201432879
[Fewster&Graham12]	Mark Fewster, Dorothy Graham, „Experiences of Test Automation: Case Studies of Software Test Automation”, Addison-Wesley 2012
[Fewster&Graham99]	Mark Fewster, Dorothy Graham, „Software Test Automation: Effective use of test execution tools”, ACM Press Books 1999, ISBN-10: 0201331403, ISBN-13: 9780201331400

[McCaffrey06]	James D. McCaffrey, „.NET Test Automation Recipes: A Problem-Solution Approach”, APRESS 2006, ISBN-13:978-1-59059-663-3, ISBN-10:1-59059-663-3
[Mosley02]	Daniel J. Mosley, Bruce A. Posey, „Just Enough Software Test Automation”, Prentice Hall 2002, ISBN-10: 0130084689, ISBN-13: 9780130084682
[Willcock11]	Colin Willcock, Thomas Deift, Stephan Tobies, Stefan Keil, „An Introduction to TTCN-3” Wiley (wydanie 2) 2011, ISBN-10: 0470663065, ISBN-13: 978-0470663066

9.5 Materiały źródłowe w Internecie

Identyfikator

ISTQB-Web

Przywoływany dokument

Serwis internetowy International Software Testing Qualifications Board, w którym są dostępne najnowsze wersje sylabusów ISTQB® www.istqb.org.

10. Uwagi dla dostawców szkoleń

10.1 Czas trwania szkolenia

Do każdego rozdziału sylabusu przypisano czas trwania w minutach. Informacja ta stanowi wskazówkę co do orientacyjnego czasu trwania poszczególnych części akredytowanego szkolenia oraz wskazuje minimalny przewidywany czas niezbędny do przekazania wiedzy zawartej w danym rozdziale.

Zajęcia prowadzone przez dostawców szkoleń mogą oczywiście trwać dłużej, a kandydaci mogą poświęcić więcej czasu na zapoznanie się z materiałem i zebranie dodatkowych informacji. Kolejność tematów w programie szkolenia nie musi pokrywać się z kolejnością wskazaną w sylabusie, a szkolenie nie musi być realizowane w formie jednego bloku zajęć.

W poniższej tabeli przedstawiono orientacyjny czas potrzebny do zaprezentowania materiału i wykonania ćwiczeń w poszczególnych rozdziałach (wszystkie wartości wyrażono w minutach).

Rozdział	Czas w minutach
0. Wstęp	0
1. Wprowadzenie do automatyzacji testów i jej cele	30
2. Przygotowanie do automatyzacji testów	165
3. Ogólna architektura testów automatycznych	270
4. Rzyzka i sytuacje awaryjne związane z wdrożeniem	150
5. Raporty i metryki dotyczące automatyzacji testów	165
6. Przeniesienie testowania manualnego do środowiska zautomatyzowanego	120
7. Weryfikowanie rozwiązania dla testów automatycznych (TAS)	120
8. Ciągłe doskonalenie	150
Razem:	1.170

Przy założeniu, że zajęcia będą prowadzone przez siedem godzin dziennie, łączny czas trwania szkolenia to 2 dni, 5 godzin i 30 minut.

10.2 Praktyczne ćwiczenia do wykonania w miejscu pracy

Nie określono żadnych ćwiczeń, które mogłyby być wykonywane w miejscu pracy.

10.3 Zasady dotyczące e-nauczania

Wszystkie części niniejszego sylabusu nadają się do wykorzystania w ramach e-nauczania.

11. Indeks

akredytacja dostawców szkoleń, 9
architektura testów automatycznych (TAA), 13
architektura testów automatycznych (TAA), 9, 23
cele biznesowe, 8
części informacyjne (egzamin), 9
części normatywne (egzamin), 9
czynniki sukcesu (automatyzacji testów), 13
diagnozowanie, 37
diagnozowanie problemów, 14, 59
egzamin, 8
ekwiwalent nakładu testowania manualnego (EMTE), 52, 54
gęstość defektów w kodzie testów automatycznych, 53, 55, 56
gTAA (generic Test Automation Architecture),
Patrz ogólna architektura testów automatycznych
interfejs wiersza poleceń (CLI), 9
inwazyjność (poziom inwazyjności), 71
inwazyjność (poziomy inwazyjności), 55
inwazyjność narzędzi, 71
ISO/IEC 25000
2014, 13
kandydaci (wymagania), 8
Konfiguracje systemu podlegającego testowaniu (SUT), 37
korzyści (z automatyzacji), 53
kryteria automatyzacji, 61
logowanie danych, 57, 58
logowanie testów, 54
łagodzenie ryzyka, 47
metryki wewnętrzne, 53
metryki zewnętrzne, 53
ocena ryzyka, 47
oczekiwania (mechanizmy), 75
ogólna architektura testów automatycznych (gTAA), 22, 24, 25
paradygmat klient-serwer, 29
paradygmat oparty na połączeniach pomiędzy pojedynczymi użytkownikami (*peer-to-peer*), 29
paradygmat oparty na zdarzeniach, 29
plik definicji testów, 34
podejście "rejestrowanie/odtworzenie", 31
podejście oparte na słowach kluczowych, 67, 75
poleceń, 61
poziom komponentów, 17
poziomy inwazyjności, 17
poziomy poznawcze (poziomy "K"), 8
projekt automatyzacji testów, 25, 27
projekt pilotażowy, 45, 46, 63
projektowanie pod kątem testowalności, 20, 71
przywrócenie sprawności (systemu), 15, 75
raportowanie, 14, 26, 27, 30, 37
rejestrowanie (danych), 14
rejestrowanie (rezultatów testów), 12, 23
rozwiązanie dla testów automatycznych (TAS), 23, 47, 58
skrót, 9
słowa kluczowe (słowa akcji), 34, 35
sterowniki, 21, 48
strategia automatyzacji testów, 14
struktura do testów automatycznych (TAF), 14
SUT (System Under Test), *Patrz* system podlegający testowaniu (SUT)
system podlegający testowaniu (SUT), 13, 17, 19, 20, 37
szacowanie, 30
śledzenie, 57, 68
śledzenie (powiązań), 67
środowisko testowe, 23, 50, 63, 65, 70
TAS, *Patrz* rozwiązanie dla testów automatycznych
technika skryptów liniowych, 3, 32, 33
technika skryptów opartych na słowach kluczowych, 34
technika skryptów sterowanych danymi, 34
technika skryptów sterowanych przez proces, 35
technika skryptów ustrukturalizowanych, 3, 33, 35
testalia, 12, 27, 50, 56
testowalność, 13, 68, 76
testowanie komponentów struktury, 71
testowanie oparte na modelu, 36
testowanie oparte na słowach kluczowych, 34, 75
testowanie potwierdzające, 40, 68
testowanie przy użyciu API, 12, 13, 14, 71
testowanie przy użyciu CLI, 12, 13
testowanie przy użyciu interfejsu GUI, 31, 32, 71
testowanie regresji, 65, 66, 68, 76, 77
testowanie sterowane danymi, 33
tworzenie skryptów, 75
utrzymywalność, 13, 75, 76
warstwa adaptacji testów, 24, 26
warstwa definiowania testów, 24, 26
warstwa wykonywania testów, 24, 26
wybór narzędzi, 18, 28
zaczep testowy (test hook), 17, 55
zarządzanie projektem, 27
zaślepki, 15, 21, 48
złożoność cyklomatyczna, 56