Certyfikowany Tester ISTQB®

sylabus Poziomu Zaawansowanego

Inżynieria Automatyzacji Testów

wersja 2.0

(w. PL 2.0.0.0)

International Software Testing Qualifications Board

Informacja o prawach autorskich

Historia zmian

Wersja	Data	Uwagi
Sylabus 2016	21.10.2016 r.	Wydanie CTAL-TAE GA
Sylabus v2.0	3.05.2024 r.	Wydanie CTAL-TAE v2.0 GA

Historia zmian wersji polskiej

Wersja	Data	Uwagi
Sylabus 2016	21.10.2016 r.	Wydanie CTAL-TAE GA
Sylabus w. 2.0 (2.0.0.0 PL)	01.10.2024 r.	Publikacja polskiej wersji sylabusa.

Spis treści

Podziękowania

Niniejszy dokument został formalnie opublikowany przez Walne Zgromadzenie ISTQB® w dniu 3 maja 2024 r.

Został on opracowany przez grupę zadaniową ds. automatyzacji testów Grupy roboczej dla poziomu Specjalista International Software Testing Qualifications Board: Graham Bath (przewodniczący Grupy roboczej dla poziomu Specjalista), Andrew Pollner (wiceprzewodniczący Grupy roboczej dla poziomu Specjalista i przewodniczący grupy zadaniowej ds. automatyzacji testów), Péter Földházi, Patrick Quilter, Gergely Ágnecz, László Szikszai. Do recenzentów grupy zadaniowej ds. automatyzacji testów należeli: Armin Beer, Armin Born, Geza Bujdoso, Renzo Cerquozzi, Jan Giesen, Arnika Hryszko, Kari Kakkonen, Gary Mogyorodi, Chris van Bael, Carsten Weise, Marc-Florian Wendland.

Recenzent techniczny: Gary Mogyorodi.

W procesie przeglądu, zgłaszania uwag i głosowania nad niniejszym sylabusem uczestniczyły następujące osoby:

Horváth Ágota, Laura Albert, Remigiusz Bednarczyk, Jürgen Beniermann, Armin Born, Alessandro Collino, Nicola De Rosa, Wim Decoutere, Ding Guofu, Istvan Forgacs, Elizabeta Fourneret, Sudhish Garg, Jan Giesen, Matthew Gregg, Tobias Horn, Mattijs Kemmink, Hardik Kori, Jayakrishnan Krishnankutty, Ashish Kulkarni, Vincenzo Marrazzo, Marton Matyas, Patricia McQuaid, Rajeev Menon, Ingvar Nordström, Arnd Pehl, Michaël Pilaeten, Daniel Polan, Nishan Portoyan, Meile Posthuma, Adam Roman, Pavel Sharikov, Péter Sótér, Lucjan Stapp, Richard Taylor, Giancarlo Tomasig, Chris Van Bael, Koen Van Belle, Johan Van Berkel, Carsten Weise, Marc-Florian Wendland, Ester Zabar.

Grupa robocza ISTQB® – Poziom Zaawansowany Inżynier Automatyzacji Testów – (wydanie 2016): Andrew Pollner (przewodniczący), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker.

0 Wstęp

0.1 Cel niniejszego sylabusa

Niniejszy sylabus stanowi podstawę egzaminu International Software Testing Qualification na Poziomie Zaawansowanym Inżynieria Automatyzacji Testów (CTAL-TAE). ISTQB® udostępnia sylabus:

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

0.2 Inżynieria automatyzacji testów w testowaniu oprogramowania

Kwalifikacja Inżynieria Automatyzacji Testów jest skierowana do wszystkich osób zaangażowanych w testowanie oprogramowania i automatyzację testów. Obejmuje to osoby na stanowiskach takich jak testerzy, analitycy testów, inżynierowie automatyzacji testów, konsultanci testów, architekci testów, kierownicy testów i programiści. Ponadto kwalifikacja ta jest odpowiednia dla osób, które pragną zdobyć podstawową wiedzę na temat automatyzacji testów, takich jak: kierownicy projektów, kierownicy ds. jakości, kierownicy ds. rozwoju oprogramowania, analitycy biznesowi, dyrektorzy IT oraz konsultanci ds. zarządzania.

Sylabus Inżynieria Automatyzacji Testów jest skierowany do inżynierów testów, którzy chcą wdrożyć lub poprawić automatyzację testów. Określono w nim metody i praktyki, które mogą wspierać zrównoważone rozwiązanie.

Inne wytyczne i modele referencyjne dotyczące rozwiązań dla testów automatycznych to standardy inżynierii oprogramowania dla wybranych cykli rozwoju oprogramowania, technologie programowania i standardy formatowania. Niniejszy sylabus nie uczy inżynierii oprogramowania. Oczekuje się jednak, że inżynier automatyzacji testów będzie dysponował umiejętnościami, doświadczeniem i wiedzą fachową w zakresie inżynierii oprogramowania.

Ponadto inżynier automatyzacji testów musi znać branżowe standardy programowania i dokumentacji oraz najlepsze praktyki, aby korzystać z nich podczas opracowywania rozwiązania dla testów automatycznych. Praktyki te mogą poprawić utrzymywalność, niezawodność i zabezpieczenia rozwiązania dla testów automatycznych. Takie standardy są zazwyczaj oparte na charakterystykach jakościowych.

0.3 Ścieżka kariery testerów i inżynierów automatyzacji testów

System stworzony przez ISTQB® umożliwia osobom zawodowo zajmującym się testowaniem dostęp do szerokiej i szczegółowej wiedzy przydatnej na każdym etapie kariery. Osoby, które uzyskają certyfikat ISTQB® w zakresie Inżynierii Automatyzacji Testów, mogą być również zainteresowane kwalifikacją w obszarze Strategii Automatyzacji Testów (CT-TAS).

Osoby, które uzyskają certyfikat ISTQB® Certyfikowany tester – Poziom Zaawansowany Inżynieria Automatyzacji Testów, mogą być również zainteresowane innymi certyfikatami w obszarze Core na Poziomie Zaawansowanym (Analityk Testów, Techniczny Analityk Testów i Zarządzanie Testami), a następnie poziomem eksperckim (Zarządzanie Testami lub Usprawnianie Procesu Testowego). Natomiast osoby chcące rozwijać swoje umiejętności w dziedzinie praktyk testowania w środowiskach zwinnych powinny wziąć pod uwagę certyfikat Techniczny Tester Zwinny lub certyfikat ATLaS (Agile Test Leadership at Scale). Strumień Specjalista udostępnia szczegółową wiedzę w obszarach, w których stosowane są określone podejścia do testowania i czynności testowe, np. Strategia Automatyzacji Testów, Tester Wydajności, Tester Zabezpieczeń, Testowanie AI i Testowanie Aplikacji Mobilnych, lub w których wymagana jest wiedza dotycząca konkretnej dziedziny (np. Tester Oprogramowania Automotive lub Testowanie Gier). Najnowsze informacje na temat systemu certyfikacji testerów ISTQB można uzyskać w serwisie www.istqb.org.

0.4 Cele biznesowe

W tej sekcji wymieniono cele biznesowe, które powinna realizować osoba, która uzyskała certyfikat Inżynieria Automatyzacji Testów.

Kandydat, który uzyskał certyfikat inżyniera automatyzacji testów, potrafi realizować następujące cele:

TAE-B01	Opisywanie celu automatyzacji testów
TAE-B02	Zrozumienie automatyzacji testów w cyklu wytwarzania oprogramowania
TAE-B03	Zrozumienie konfiguracji infrastruktury umożliwiającej automatyzację testów
TAE-B04	Znajomość procesu oceny w celu wyboru odpowiednich narzędzi i strategii
TAE-B05	Zrozumienie koncepcji projektowania w zakresie budowania modułowych i skalowalnych rozwiązań dla testów automatycznych
TAE-B06	Wybór podejścia, w tym pilotażowego, w celu planowania wdrożenia automatyzacji testów w cyklu wytwarzania oprogramowania
TAE-B07	Projektowanie i opracowywanie (nowych lub zmodyfikowanych) rozwiązań dla testów automatycznych, które odpowiadają potrzebom technicznym
TAE-B08	Rozważenie zakresu i podejścia do automatyzacji testów i pielęgnacji testaliów
TAE-B09	Zrozumienie, w jaki sposób testy automatyczne integrują się z potokami CI/CD
TAE-B10	Zrozumienie, w jaki sposób gromadzić, analizować i raportować dane dotyczące automatyzacji testów w celu informowania interesariuszy
TAE-B11	Weryfikacja infrastruktury testów automatycznych
TAE-B12	Określanie możliwości ciągłego doskonalenia automatyzacji testów

0.5 Cele nauczania objęte egzaminem i poziomy wiedzy

Cele nauczania wspomagają osiąganie celów biznesowych i są wykorzystywane do opracowywania egzaminów certyfikacyjnych w zakresie certyfikacji Certyfikowany Tester Inżynieria Automatyzacji Testów.

Ogólnie rzecz biorąc, wszystkie treści tego sylabusa można objąć egzaminem na poziomach K2, K3 i K4, z wyjątkiem wstępu i załączników. Oznacza to, że od kandydata może być wymagane rozpoznanie, zapamiętanie lub przypomnienie sobie słowa kluczowego lub pojęcia wymienionego w każdym z ośmiu rozdziałów dokumentu. Poziomy wiedzy w odniesieniu do poszczególnych celów nauczania przedstawiono na początku każdego rozdziału. Poziomy te sklasyfikowano następująco:

- K2: Zrozumieć
- K3: Zastosować
- K4: Przeanalizować

Dalsze szczegóły i przykłady celów nauczania podano w Załączniku A.

Definicje wszystkich pojęć wymienionych jako słowa kluczowe pod tytułami rozdziałów należy zapamiętać, nawet jeśli nie wspomniano o tym wyraźnie w celach nauczania.

0.6 Egzamin certyfikacyjny z zakresu inżynierii automatyzacji testów

Egzamin z zakresu inżynierii automatyzacji testów będzie oparty na tym sylabusie. Przy udzielaniu odpowiedzi na pytania egzaminacyjne może być konieczne skorzystanie z materiału obejmującego więcej niż jeden rozdział tego sylabusa. Przedmiotem egzaminu może być treść wszystkich części sylabusa z wyjątkiem wstępu i załączników. W dokumencie znajdują się również odwołania do norm/standardów i książek, ale ich treść nie może być przedmiotem egzaminu w zakresie wykraczającym poza informacje streszczone w tym sylabusie.

Więcej informacji na temat egzaminu z zakresu inżynierii automatyzacji testów można znaleźć w dokumencie „Exam Structures and Rules v1.1 Compatible with Syllabus Foundation and Advanced Levels and Specialists Modules”.

Kryterium przystąpienia do egzaminu w zakresie inżynierii automatyzacji testów jest to, że kandydaci są zainteresowani testowaniem oprogramowania i automatyzacją testów. Zdecydowanie zaleca się jednak, aby kandydaci:

- mieli przynajmniej minimalne doświadczenie w wytwarzaniu i testowaniu oprogramowania, np. sześciomiesięczne doświadczenie jako inżynier testów oprogramowania lub jako programista;
- wzięli udział w kursie, który został akredytowany zgodnie ze standardami ISTQB® (przez jedną z Rad Krajowych uznanych przez ISTQB®).

Uwaga dotycząca wymogów wstępnych: przed przystąpieniem do egzaminu certyfikacyjnego ISTQB® Inżynieria Automatyzacji Testów należy uzyskać certyfikat ISTQB® Certyfikowany Tester Poziom Podstawowy.

0.7 Akredytacja

Rada Krajowa ISTQB® może dokonywać akredytacji dostawców szkoleń, którzy oferują materiały dydaktyczne zgodne z niniejszym sylabusem. Wytyczne dotyczące akredytacji należy uzyskać od Rady Krajowej lub organu dokonującego akredytacji. Akredytowane szkolenie jest uznawane za zgodne z niniejszym sylabusem i może obejmować egzamin ISTQB®.

Wytyczne dotyczące akredytacji w zakresie niniejszego sylabusa są zgodne z ogólnymi wytycznymi dotyczącymi akredytacji opublikowanymi przez Grupę roboczą ds. zarządzania procesami i zgodności.

0.8 Odniesienia do norm

W sylabusie Inżynieria Automatyzacji Testów znajdują się odniesienia do norm (np. IEEE i ISO). Celem tych odniesień jest zapewnienie ram (jak w odniesieniach do ISO 25010 dotyczących charakterystyk jakościowych) lub zapewnienie źródła dodatkowych informacji, jeśli czytelnik tego potrzebuje. Należy pamiętać, że w sylabusie wykorzystano standardowe dokumenty jako odniesienie. Należy jednak zaznaczyć, że treść norm i standardów nie jest przedmiotem egzaminu. Więcej informacji na temat norm można znaleźć w rozdziale 9 Bibliografia.

0.9 Ciągła aktualizacja

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

0.10 Poziom szczegółowości

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

- ogólne cele instruktażowe opisujące założenia Inżynierii Automatyzacji Testów;
- wykaz terminów, które muszą zapamiętać uczestnicy szkolenia;
- cele nauczania w poszczególnych obszarach wiedzy, opisujące efekty kształcenia o charakterze poznawczym;
- opis kluczowych pojęć, w tym odniesienia do źródeł, takich jak przyjęta literatura lub normy.

Treść sylabusa nie stanowi opisu całego obszaru wiedzy związanego z testowaniem oprogramowania. Odzwierciedla ona jedynie poziom szczegółowości, jaki należy uwzględnić w szkoleniach z zakresu inżynierii automatyzacji testów. Skupia się na pojęciach i technikach związanych z testowaniem, które mogą mieć zastosowanie do wszystkich projektów związanych z oprogramowaniem, w tym do tych związanych z metodykami zwinnymi. W niniejszym sylabusie nie zawarto żadnych konkretnych celów nauczania związanych z testowaniem zwinnym, ale omówiono to, w jaki sposób koncepcje te mają zastosowanie w projektach zwinnych i innych rodzajach projektów.

0.11 Struktura sylabusa

Sylabus składa się z ośmiu rozdziałów zawierających treści będące przedmiotem egzaminu. Nagłówek najwyższego poziomu każdego rozdziału określa czas na dany rozdział; czas nie jest podany poniżej poziomu rozdziału. W przypadku akredytowanych szkoleń sylabus wymaga co najmniej 21 godzin zajęć, podzielonych na osiem rozdziałów w następujący sposób:

- Rozdział 1: 45 minut – wprowadzenie i cele automatyzacji testów
  - Tester dowiaduje się o zaletach automatyzacji testów i jej ograniczeniach;
  - Omawiana jest automatyzacja testów w różnych modelach cyklu wytwarzania oprogramowania;
  - Tester dowiaduje się, w jaki sposób architektura systemu podlegającego testowaniu (SUT) wpływa na przydatność narzędzi testowych.
- Rozdział 2: 180 minut – przygotowanie do automatyzacji testów
  - Omawiane jest projektowanie pod kątem testowalności SUT poprzez obserwowalność, sterowalność i jasno określoną architekturę;
  - Tester dowiaduje się o automatyzacji testów w różnych środowiskach;
  - Omawiane są czynniki wymagane do oceny odpowiedniego rozwiązania dla testów automatycznych;
  - Tester zapoznaje się z czynnikami technicznymi potrzebnymi do opracowania zaleceń dotyczących automatyzacji testów.
- Rozdział 3: 210 minut – architektura testów automatycznych
  - Omawiana jest architektura testów automatycznych i jej komponenty prowadzące do rozwiązania dla testów automatycznych;
  - Tester zapoznaje się z warstwami i ich zastosowaniem w ramach struktury do testów automatycznych;
  - Omawiane są różne podejścia do korzystania z narzędzi do automatyzacji testów;
  - Tester dowiaduje się, w jaki sposób zasady projektowania i wzorce projektowe mogą być stosowane do automatyzacji testów.
- Rozdział 4: 150 minut – wdrażanie automatyzacji testów
  - Omawiane są sposoby skutecznego planowania i wdrażania projektu pilotażowego w zakresie automatyzacji testów;
  - Tester zapoznaje się z zagrożeniami związanymi z wdrażaniem i strategiami łagodzenia ich skutków;
  - Omawiane są czynniki poprawiające utrzymywalność kodu automatyzacji testów.
- Rozdział 5: 90 minut – wdrażanie i strategie wdrażania automatyzacji testów
  - Tester zapoznaje się z potokami CI/CD i wykonywaniem zautomatyzowanych testów na różnych poziomach testów;
  - Omawiane jest zarządzanie konfiguracją komponentów automatyzacji testów;
  - Tester zapoznaje się z zależnościami stosowanymi w odniesieniu do API i testowania kontraktów.
- Rozdział 6: 150 minut – raportowanie i metryki automatyzacji testów
  - Tester dowiaduje się, gdzie można gromadzić dane z SUT i z testów automatycznych w celu ich analizy i raportowania;
  - Omawiane są analizy danych z raportów SUT i automatyzacji testów w celu wykrywania przyczyn awarii;
  - Omawiane są raporty z testów i tablice wskaźników, które mają na celu informowanie interesariuszy.

- Rozdział 7: 135 minut – weryfikacja rozwiązania dla testów automatycznych
  - Tester uczy się, jak badać i weryfikować poprawność działania komponentów i środowiska automatyzacji testów;
  - Omawiany jest sposób zapewnienia prawidłowego działania skryptów i pakietów testowych;
  - Tester dowiaduje się, kiedy należy przeprowadzać analizę przyczyny podstawowej;
  - Omawiane są techniki analizy kodu automatyzacji testów pod kątem jakości.
- Rozdział 8: 210 minut – ciągłe doskonalenie
  - Omawiane są dodatkowe obszary analizy danych w celu poprawy przypadków testowych;
  - Tester zapoznaje się ze sposobami wprowadzania udoskonaleń i uaktualnień do rozwiązania dla testów automatycznych i jego komponentów;
  - Omawiane jest określanie sposobów konsolidacji i usprawnienia automatyzacji testów;
  - Tester dowiaduje się, w jaki sposób narzędzia do automatyzacji testów mogą pomóc w obsłudze testów i potrzebach związanych z konfiguracją.

1 Wprowadzenie i cele automatyzacji testów – 45 minut (K-2)

Słowa kluczowe

automatyzacja testów, inżynier automatyzacji testów, system podlegający testowaniu

Cele nauczania w rozdziale 1:

1.1 Cel automatyzacji testów

TAE-1.1.1 (K2) Kandydat wyjaśnia zalety i wady automatyzacji testów.

1.2 Automatyzacja testów w cyklu wytwarzania oprogramowania

TAE-1.2.1 (K2) Kandydat wyjaśnia, w jaki sposób automatyzacja testów jest stosowana w różnych modelach cyklu wytwarzania oprogramowania.

TAE-1.2.2 (K2) Kandydat wybiera odpowiednie narzędzia do automatyzacji testów w odniesieniu do danego systemu podlegającego testowaniu.

1.1 Cel automatyzacji testów

1.1.1 Kandydat wyjaśnia zalety i wady automatyzacji testów

Automatyzacja testów, która obejmuje automatyczne wykonywanie testów i raportowanie testów, to co najmniej jedno z następujących działań:

- korzystanie z dedykowanych narzędzi programowych do sterowania i konfigurowania zestawów testowych do wykonywania testów;
- wykonywanie testów w sposób zautomatyzowany;
- porównywanie rzeczywistych wyników z oczekiwanymi.

Automatyzacja testów zapewnia znaczące funkcje (cechy) i możliwości, które mogą wchodzić w interakcję z systemem podlegającym testowaniu (ang. *System Under Test* - SUT). Automatyzacja testów może obejmować szeroki zakres oprogramowania. Rozwiązania obejmują wiele rodzajów oprogramowania (np. SUT z interfejsem użytkownika (UI), SUT bez interfejsu użytkownika, aplikacje mobilne, protokoły sieciowe i połączenia).

Automatyzacja testów ma wiele zalet:

- umożliwia przeprowadzenie większej liczby testów na kompilację w porównaniu z testami manualnymi;
- zapewnia możliwość tworzenia i wykonywania testów, których nie można wykonać manualnie (np. reakcja w czasie rzeczywistym, testowanie zdalne i testowanie równoległe);
- umożliwia wykonywanie testów, które są bardziej złożone niż testy manualne;
- wykonuje testy szybciej niż testowanie manualne;
- jest w mniejszym stopniu wrażliwa na błędy ludzkie;
- bardziej skutecznie i wydajnie korzysta z zasobów testowych;
- zapewnia szybsze informacje zwrotne dotyczące jakości SUT;
- poprawia niezawodność systemu (np. dostępność i odtwarzalność);
- poprawia spójność wykonywania testów w różnych cyklach testowych.

Automatyzacja testów ma jednak potencjalne wady, w tym:

- projekt będzie wiązał się z dodatkowymi kosztami, ponieważ może zaistnieć potrzeba zatrudnienia inżyniera automatyzacji testów (TAE), zakupu nowego sprzętu i zorganizowania szkoleń;
- wymóg inwestycji początkowej w celu skonfigurowania rozwiązania dla testów automatycznych;
- czas na opracowanie i utrzymanie rozwiązania dla testów automatycznych;
- wymóg jasnych celów automatyzacji testów w celu zapewnienia sukcesu;
- sztywność testów i mniejsza zdolność adaptacji do zmian w SUT;
- wprowadzanie dodatkowych defektów przez automatyzację testów.

Istnieją ograniczenia dotyczące automatyzacji testów, o których należy pamiętać:

- nie wszystkie testy manualne można zautomatyzować;
- weryfikuje tylko to, do czego zaprogramowane zostały testy automatyczne;
- automatyzacja testów może sprawdzać tylko wyniki testów, które można interpretować maszynowo, co oznacza, że niektóre charakterystyki jakościowe mogą nie być możliwe do przetestowania za pomocą automatyzacji;
- automatyzacja testów może sprawdzać tylko wyniki testów, które mogą być zweryfikowane przez zautomatyzowaną wyrocznię testową.

1.2 Automatyzacja testów w cyklu wytwarzania oprogramowania

1.2.1 Kandydat wyjaśnia, w jaki sposób automatyzacja testów jest stosowana w różnych modelach cyklu wytwarzania oprogramowania

Model kaskadowy

Model kaskadowy jest modelem cyklu wytwarzania oprogramowania (SDLC), który jest zarówno liniowy, jak i sekwencyjny. Model ten ma różne fazy (tj. wymagania, projekt, wdrożenie, weryfikacja i pielęgnacja), a każda faza zazwyczaj kończy się dokumentacją, która musi zostać zatwierdzona. Implementacja automatyzacji testów zwykle odbywa się równolegle lub po fazie implementacji. Przebiegi testów zwykle odbywają się podczas fazy weryfikacji ze względu na to, że komponenty oprogramowania nie były gotowe do testowania do tego czasu.

Model V

Model V jest modelem SDLC, w którym proces jest wykonywany w sposób sekwencyjny. Ponieważ projekt jest definiowany od wymagań wysokiego poziomu do wymagań niskiego poziomu, odpowiednie czynności testowe i integracyjne są definiowane w celu walidacji tych wymagań. Tradycyjne poziomy testów to: testowanie modułowe, testowanie integracji modułów, testowanie systemowe, testowanie integracji systemów, testowanie akceptacyjne (jak opisano w podrozdziale 2.2 CTFL). Możliwe i zalecane jest zapewnienie struktury do testów automatycznych (ang. *Test Automation Framework* - TAF) w odniesieniu do każdego poziomu testów.

Zwinne wytwarzanie oprogramowania

W zwinnym wytwarzaniu oprogramowania istnieje wiele możliwości automatyzacji testów. W przeciwieństwie do modelu kaskadowego lub modelu V, w metodyce zwinnego wytwarzania oprogramowania inżynierowie automatyzacji testów (TAEs) i przedstawiciele biznesu mogą decydować o planie, harmonogramie i planowanej dostawie testów. W tej metodyce istnieją najlepsze praktyki, takie jak przeglądy kodu, programowanie w parach i częste wykonywanie testów automatycznych. Eliminacja silosów (tj. upewnienie się, że programiści, testerzy i inni interesariusze współpracują ze sobą) pozwala zespołom objąć wszystkie poziomy testów za pomocą odpowiedniej ilości i głębokości automatyzacji testów, aby osiągnąć cel zwany automatyzacją w sprincie. Więcej szczegółów można znaleźć w sylabusie ISTQB CT-TAS, podrozdział 3.2.

1.2.2 Kandydat wybiera odpowiednie narzędzia do automatyzacji testów w odniesieniu do danego systemu podlegającego testowaniu

Aby określić najbardziej odpowiednie narzędzia testowe na potrzeby danego projektu, należy najpierw przeanalizować SUT. TAE muszą określić wymagania projektowe, które mogą być wykorzystane jako punkt odniesienia przy wyborze narzędzi.

Ponieważ inne funkcje narzędzi do automatyzacji testów są używane w oprogramowaniu UI i, na przykład, w usługach internetowych (ang. *webservices*), ważne jest, aby zrozumieć, co projekt ma za zadanie osiągnąć w czasie. Nie ma ograniczeń co do liczby narzędzi do automatyzacji testów i funkcji, które mogą być używane lub wybierane, ale koszty muszą być zawsze brane pod uwagę. Korzystanie z gotowego narzędzia komercyjnego lub wdrożenie niestandardowego rozwiązania opartego na technologii open source może być złożonym procesem.

Kolejnym tematem do oceny jest skład i doświadczenie zespołu w zakresie automatyzacji testów. W przypadku, gdy testerzy mają niewielkie doświadczenie w programowaniu lub nie mają go wcale, dobrym wyborem może być użycie rozwiązania niskokodowego lub bezkodowego.

W przypadku testerów technicznych, którzy dysponują wiedzą z zakresu programowania, pomocny może być wybór narzędzi, których język odpowiada językowi SUT. Zapewnia to korzyści, w tym możliwość bardziej efektywnej pracy z programistami nad debugowaniem defektów testów automatycznych i wzajemnym szkoleniem członków między zespołami.

2 Przygotowanie do automatyzacji testów – 180 minut (K4)

Słowa kluczowe

testowalność, testowanie API, testowanie GUI

Cele nauczania w rozdziale 2:

2.1 Zrozumienie konfiguracji infrastruktury umożliwiającej automatyzację testów

TAE-2.1.1 (K2) Kandydat opisuje potrzeby konfiguracyjne infrastruktury umożliwiające wdrożenie automatyzacji testów.

TAE-2.1.2 (K2) Kandydat wyjaśnia, w jaki sposób automatyzacja testów jest wykorzystywana w różnych środowiskach.

2.2 Proces oceny w celu wyboru odpowiednich narzędzi i strategii

TAE-2.2.1 (K4) Kandydat analizuje system podlegający testowaniu w celu określenia odpowiedniego rozwiązania dla testów automatycznych.

TAE-2.2.2 (K4) Kandydat ilustruje wyniki techniczne oceny narzędzia.

2.1 Zrozumienie konfiguracji infrastruktury umożliwiającej automatyzację testów

2.1.1 Kandydat opisuje potrzeby konfiguracyjne infrastruktury umożliwiające wdrożenie automatyzacji testów

Testowalność SUT (tj. dostępność interfejsów programowych obsługujących testowanie, np. w celu umożliwienia kontroli i obserwowalności SUT) powinna być zaprojektowana i wdrożona równolegle z projektowaniem i wdrożeniem innych funkcji SUT. Ta praca jest zwykle wykonywana przez architekta oprogramowania, ponieważ testowalność jest niefunkcjonalnym wymaganiem systemu, często z udziałem TAE, w celu zidentyfikowania konkretnych obszarów, w których można wprowadzić udoskonalenia.

W celu zapewnienia lepszej testowalności SUT można wykorzystać różne rozwiązania, które mają różne potrzeby konfiguracyjne, na przykład:

- Identyfikatory dostępności
  - Różne struktury programistyczne mogą generować te identyfikatory automatycznie lub programiści mogą je ustawić manualnie
- Zmienne środowiska systemowego
  - Niektóre parametry aplikacji można zmienić, aby umożliwić łatwiejsze testowanie poprzez administrację
- Zmienne wdrażania
  - Podobne do zmiennych systemowych, ale można je skonfigurować przed rozpoczęciem wdrażania

Projektowanie pod kątem testowalności SUT obejmuje następujące aspekty:

- Obserwowalność: SUT musi zapewniać interfejsy, które dają wgląd w SUT. Przypadki testowe mogą następnie wykorzystać te interfejsy do ustalenia, czy rzeczywiste wyniki są równe oczekiwanym wynikom.
- Sterowalność: SUT musi zapewniać interfejsy, które można wykorzystać do wykonywania działań na SUT. Mogą to być elementy interfejsu użytkownika, wywołania funkcji, elementy komunikacyjne (np. protokoły Transmission Control Protocol/Internet Protocol (TCP/IP) i Universal Serial Bus (USB)) lub sygnały elektroniczne dla przełączników fizycznych lub logicznych w różnych zmiennych środowiskowych.
- Przejrzystość architektury: dokumentacja architektury musi zapewniać jasne, zrozumiałe komponenty i interfejsy, które zapewniają obserwowalność i sterowalność na wszystkich poziomach testów oraz sprzyjają jakości.

2.1.2 Kandydat wyjaśnia, w jaki sposób automatyzacja testów jest wykorzystywana w różnych środowiskach

W różnych środowiskach mogą być wykonywane różne typy testów automatycznych. Środowiska te mogą różnić się między projektami i metodykami, a większość projektów ma jedno lub więcej środowisk do wykorzystania do testowania. Z technicznego punktu widzenia środowiska te można tworzyć z kontenerów, oprogramowania do wirtualizacji i przy użyciu innych podejść.

Zestaw możliwych do rozważenia środowisk obejmuje następujące elementy:

Lokalne środowisko programistyczne

Lokalne środowisko programistyczne to miejsce, w którym początkowo tworzone jest oprogramowanie, a moduły są testowane za pomocą automatyzacji w celu weryfikacji przydatności funkcjonalnej. W lokalnym środowisku programistycznym może wystąpić kilka typów testów, w tym testowanie modułowe, testowanie GUI i testowanie interfejsu programowania aplikacji (API). Należy również pamiętać, że przy użyciu zintegrowanego środowiska programistycznego (IDE) na danym komputerze można wcześnie przeprowadzić testowanie białoskrzynkowe, aby zidentyfikować słabe kodowanie i problemy z jakością.

Środowisko kompilacji

Jego głównym celem jest budowanie oprogramowania i wykonywanie testów, które sprawdzają poprawność powstałej kompilacji w ekosystemie DevOps. Środowisko to może być lokalnym środowiskiem programistycznym lub agentem ciągłej integracji/ciągłego dostarczania (CI/CD), w którym można przeprowadzać testy niskiego poziomu (tj. testy modułowe i testy integracji modułów) oraz analizę statyczną bez rzeczywistego wdrażania do jakichkolwiek innych środowisk.

Środowisko integracji

Po wykonaniu testów niskopoziomowych i analizie statycznej kolejnym etapem jest środowisko integracji systemów. W tym przypadku obecny jest kandydat do wydania SUT, w pełni zintegrowany z innymi systemami, które mogą być testowane. W tym środowisku można wykonać w pełni zautomatyzowany zestaw testowy, zarówno testy interfejsu użytkownika, jak i testy interfejsu API. W tym środowisku nie ma testów białoskrzynkowych, tylko testy czarnoskrzynkowe (tj. integracja systemu i/lub testowanie akceptacyjne). Należy pamiętać, że jest to pierwsze środowisko, w którym powinno być obecne monitorowanie, aby zobaczyć, co dzieje się w tle podczas korzystania z SUT, co umożliwi skuteczne badanie defektów/awarii.

Środowisko przedprodukcyjne

Środowisko przedprodukcyjne jest używane głównie do oceny niefunkcjonalnych charakterystyk jakościowych (np. wydajności). Mimo że testy niefunkcjonalne można przeprowadzać w dowolnym środowisku, szczególny nacisk kładzie się na środowisko przedprodukcyjne, ponieważ przypomina ono środowisko produkcyjne tak bardzo, jak to możliwe. Często testowanie akceptacyjne przez użytkownika może być wykonywane przez interesariuszy biznesowych w celu weryfikacji produktu końcowego, a w razie potrzeby możliwe jest również wykonanie istniejącego zautomatyzowanego zestawu testowego. To środowisko również jest monitorowane.

Środowisko produkcyjne/operacyjne

Środowisko produkcyjne może być używane do oceny funkcjonalnych i niefunkcjonalnych charakterystyk jakościowych w czasie rzeczywistym, podczas gdy użytkownicy wchodzą w interakcję z wdrożonym systemem z monitorowaniem i pewnymi najlepszymi praktykami, które umożliwiają testowanie w środowisku produkcyjnym (np. testy kanarkowe, wdrożenie niebiesko-zielone (ang. *blue/green deployment*) i testowanie A/B).

2.2 Proces oceny w celu wyboru odpowiednich narzędzi i strategii

2.2.1 Kandydat analizuje system podlegający testowaniu w celu określenia odpowiedniego rozwiązania dla testów automatycznych

Wszystkie SUT mogą się różnić między sobą, ale istnieją różne czynniki i cechy, które można przeanalizować, aby uzyskać skuteczne rozwiązanie dla testów automatycznych (ang. *Test Automation Solution* - TAS). Podczas badania SUT TAE muszą zebrać wymagania, biorąc pod uwagę jego zakres i istniejące możliwości. Różne rodzaje aplikacji (np. usługi internetowe, aplikacje mobilne i webowe) z technicznego punktu widzenia wymagają różnych rodzajów automatyzacji testów. Badanie może być przeprowadzone – i jest zalecane – we współpracy z innymi interesariuszami (np. testerami manualnymi, interesariuszami biznesowymi i analitykami biznesowymi) w celu zidentyfikowania jak największej liczby ryzyk i ich łagodzenia, aby uzyskać korzystne rozwiązanie dla testów automatycznych na przyszłość.

Wymagania dotyczące podejścia i architektury automatyzacji testów powinny uwzględniać następujące zagadnienia:

- Jakie czynności procesu testowego powinny być zautomatyzowane (np. zarządzanie testami, projektowanie testów, generowanie testów i wykonywanie testów);
- Jakie poziomy testów powinny być obsługiwane;
- Jakie typy testów powinny być obsługiwane;
- Jakie role testowe i zestawy umiejętności powinny być obsługiwane;
- Jakie produkty oprogramowania, linie produktów i rodziny produktów powinny być obsługiwane (np. w celu określenia zakresu i żywotności wdrożonego TAS);
- Jakie rodzaje SUT muszą być zgodne z TAS;
- Dostępność danych testowych i ich jakość;
- Możliwe metody i sposoby emulowania nieosiągalnych przypadków (np. zaangażowane aplikacje innych firm).

2.2.2 Kandydat ilustruje wyniki techniczne oceny narzędzia

Po przeanalizowaniu SUT i zebraniu wymagań od wszystkich interesariuszy prawdopodobnie będą istnieć narzędzia do automatyzacji testów, które będą odpowiadać tym wymaganiom i które będzie można rozważyć. Może nie istnieć jedno narzędzie, które spełniałoby wszystkie zidentyfikowane wymagania, a interesariusze powinni brać pod uwagę tę możliwość.

Pomocne jest uwzględnienie ustaleń dotyczących możliwych narzędzi i zastanowienie się nad różnymi bezpośrednimi i pośrednimi wymaganiami w tabeli porównawczej. Celem tabeli porównawczej jest umożliwienie interesariuszom dostrzeżenia różnic między narzędziami w oparciu o określone wymagania. Tabela porównawcza zawiera listę narzędzi w kolumnach oraz wymagania w wierszach. Komórki zawierają informacje o właściwościach każdego narzędzia w odniesieniu do każdego wymagania i priorytetów.

Ogólnie rzecz biorąc, narzędzia do automatyzacji testów powinny być oceniane w celu ustalenia, czy spełniają one wymagania określone w poprzedniej sekcji (2.2.1). Wymagania, które należy wziąć pod uwagę podczas oceny i porównywania narzędzi:

- język/technologia narzędzia oraz narzędzia IDE;
- możliwość konfiguracji narzędzia, niezależnie od tego, czy obsługuje ono różne środowiska testowe, uruchamia konfiguracje i wykorzystuje dynamiczne lub statyczne wartości konfiguracji;

- możliwość zarządzania danymi testowymi w narzędziu. Zarządzanie danymi testowymi może być zintegrowane z centralnym repozytorium kontroli wersji;
- w przypadku różnych typów testów może być konieczne wybranie różnych narzędzi do automatyzacji testów;
- możliwość zapewnienia raportowania. Jest to ważne w celu dostosowania się do wymagań dotyczących raportowania testów w projekcie;
- możliwość integracji z innymi narzędziami używanymi w projekcie lub organizacji, takimi jak CI/CD, śledzenie zadań, zarządzanie testami, raportowanie lub inne narzędzia;
- możliwość rozszerzenia ogólnej architektury testów i oceny skalowalności, utrzymywalności, modyfikowalności, kompatybilności i niezawodności narzędzi.

Ta tabela porównawcza jest dobrym źródłem do określenia proponowanego narzędzia lub zestawu narzędzi do wykorzystania do automatyzacji testów SUT.

Proces może się różnić w zależności od tego, w jaki sposób podejmowana jest decyzja o tym, które narzędzie(-a) zostanie(-ą) użyte, ale propozycja powinna zostać przedstawiona odpowiednim interesariuszom do zatwierdzenia.

3 Architektura testów automatycznych – 210 minut (K3)

Słowa kluczowe

jarzmo testowe, krok testowy, ogólna architektura testów automatycznych, rejestrowanie-odtwarzanie, rozwiązanie dla testów automatycznych, skrypt testowy, skrypty liniowe, struktura do testów automatycznych, technika skryptów ustrukturalizowanych, testalia, testowanie oparte na modelu, testowanie sterowane danymi, testowanie sterowane słowami kluczowymi, warstwa adaptacji testów, wytwarzanie sterowane testami, wytwarzanie sterowane zachowaniem

Cele nauczania w rozdziale 3:

3.1 Koncepcje projektowe wykorzystywane w automatyzacji testów

TAE-3.1.1 (K2) Kandydat wyjaśnia główne możliwości w architekturze testów automatycznych.

TAE-3.1.2 (K2) Kandydat wyjaśnia, jak zaprojektować rozwiązanie dla testów automatycznych.

TAE-3.1.3 (K3) Kandydat stosuje warstwowanie struktur do testów automatycznych.

TAE-3.1.4 (K3) Kandydat stosuje różne podejścia do automatyzacji przypadków testowych.

TAE-3.1.5 (K3) Kandydat stosuje zasady projektowania i wzorce projektowe w automatyzacji testów.

3.1 Koncepcje projektowe wykorzystywane w automatyzacji testów

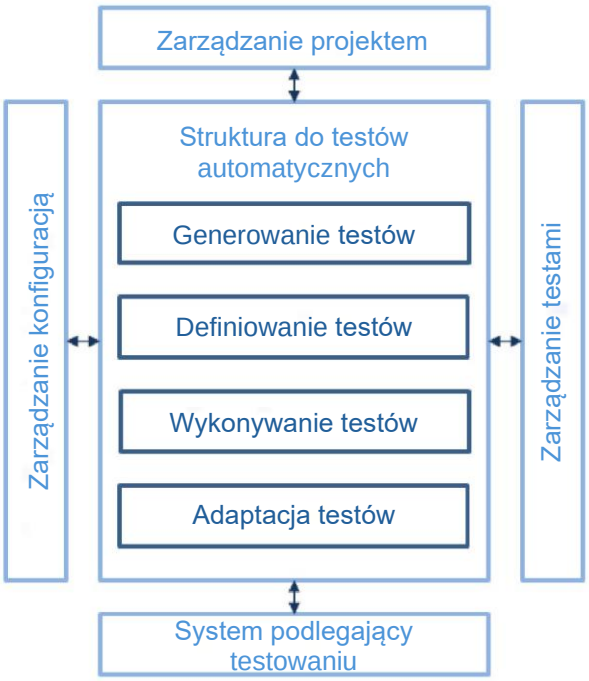
3.1.1 Kandydat wyjaśnia główne możliwości w architekturze testów automatycznych

Ogólna architektura testów automatycznych (gTAA)

gTAA to koncepcja projektowania wysokiego poziomu, która zapewnia abstrakcyjny widok komunikacji między testami automatycznymi a systemami, do których podłączone są testy automatyczne, tj. SUT, zarządzanie projektami, zarządzanie testami i zarządzanie konfiguracją (patrz Rysunek 1). Zapewnia ona również możliwości, które są niezbędne do pokrycia podczas projektowania architektury testów automatycznych (TAA).

Interfejsy gTAA opisują następujące elementy:

- Interfejs SUT opisuje połączenie między SUT a TAF (patrz sekcja 3.1.3 dotycząca struktury do testów automatycznych).
- Interfejs zarządzania projektem opisuje postęp prac nad wdrożeniem automatyzacji testów.
- Interfejs zarządzania testami opisuje mapowanie definicji przypadków testowych i zautomatyzowanych przypadków testowych.
- Interfejs zarządzania konfiguracją opisuje potoki CI/CD, środowiska i testalia.



Rysunek 1. Schemat gTAA

Możliwości oferowane przez biblioteki i narzędzia do automatyzacji testów

Podstawowe możliwości automatyzacji testów powinny być identyfikowane i wybierane spośród dostępnych narzędzi zgodnie z wymaganiami danego projektu.

Generowanie testów: obsługuje automatyczne projektowanie przypadków testowych w oparciu o model testowy. Narzędzia do testowania opartego na modelu można wykorzystać w procesie generowania (patrz sylabus ISTQB CT-MBT). Generowanie testów jest możliwością opcjonalną.

Definiowanie testów: obsługuje definiowanie i implementację przypadków testowych i/lub zestawów testowych, które opcjonalnie można uzyskać z modelu testowego. Oddziela definicję testu od SUT i/lub

narzędzi testowych. Zawiera środki do definiowania testów wysokiego i niskiego poziomu, które są obsługiwane w danych testowych, przypadkach testowych i komponentach biblioteki testowej lub ich kombinacjach.

Wykonywanie testów: obsługuje wykonywanie testów i rejestrowanie testów. Zapewnia narzędzie do wykonywania testów służące do automatycznego uruchamiania wybranych testów oraz komponent do rejestrowania testów i raportowania testów.

Adaptacja testów: zapewnia niezbędną funkcjonalność do dostosowywania testów automatycznych do różnych modułów lub interfejsów SUT. Zapewnia różne adaptery do łączenia się z SUT za pośrednictwem interfejsów API, protokołów i usług.

3.1.2 Kandydat wyjaśnia, jak zaprojektować rozwiązanie dla testów automatycznych

Rozwiązanie dla testów automatycznych (TAS) jest definiowane przez zrozumienie wymagań funkcjonalnych, niefunkcjonalnych i technicznych SUT, istniejących lub wymaganych narzędzi, które są niezbędne do wdrożenia rozwiązania. TAS jest wdrażane za pomocą narzędzi komercyjnych lub *open source* i może wymagać dodatkowych adapterów specyficznych dla SUT.

Architektura TAA definiuje projekt techniczny dla całego TAS. Powinna określać:

- wybór narzędzi do automatyzacji testów i bibliotek specyficznych dla narzędzi;
- opracowywanie wtyczek i/lub komponentów;
- identyfikowanie wymagań dotyczących łączności i interfejsów (np. zapory sieciowe, baza danych, jednolite lokalizatory zasobów (URL)/połączenia, makiety/zaślepki, kolejki wiadomości i protokoły);
- łączenie się z narzędziami do zarządzania testami i zarządzania defektami;
- wykorzystanie systemu kontroli wersji i repozytoriów.

3.1.3 Kandydat stosuje warstwowanie struktur do testów automatycznych

Struktura do testów automatycznych (TAF)

TAF jest podstawą TAS. Często zawiera jarzmo testowe, biblioteki testowe, skrypty testowe i zestawy testowe.

Warstwy TAF

Warstwy TAF definiują wyraźną granicę klas, które mają podobne cele, takie jak przypadki testowe, raportowanie testów, rejestrowanie testów, szyfrowanie i jarzma testowe. Poprzez wprowadzenie warstwy do każdego pojedynczego celu, projekt może stać się skomplikowany. W związku z tym zaleca się utrzymywanie małej liczby warstw TAF.

Skrypty testowe

Ich celem jest zapewnienie repozytorium przypadków testowych SUT i adnotacji zestawu testowego. Wywołują usługi warstwy logiki biznesowej, które mogą obejmować kroki testowe, przepływy użytkowników lub wywołania API. Nie należy jednak wykonywać bezpośrednich wywołań do podstawowych bibliotek ze skryptów testowych.

Rysunek 2. Warstwy struktury do testów automatycznych

Logika biznesowa

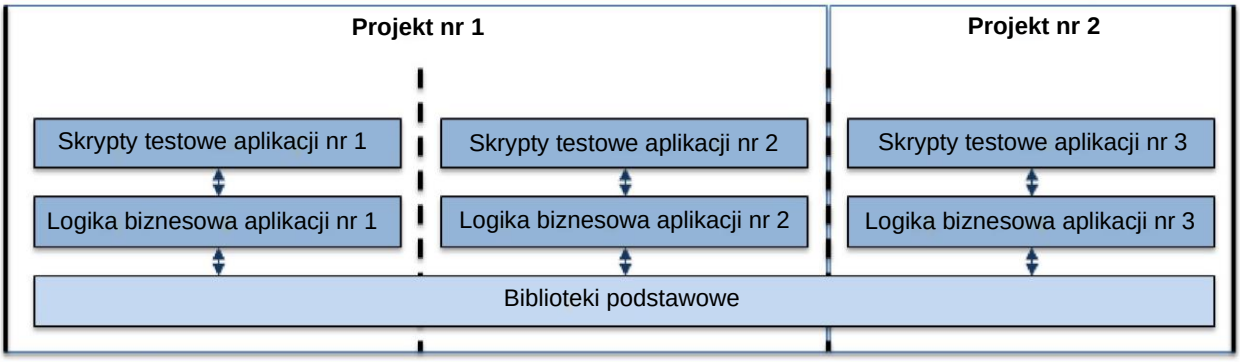
W tej warstwie przechowywane są wszystkie biblioteki zależne od SUT. Biblioteki te będą dziedziczyć pliki klas bibliotek podstawowych lub korzystać z dostarczonych przez nie fasad (patrz sekcja 3.1.5 dotycząca dziedziczenia i fasad). Warstwa logiki biznesowej służy do konfigurowania TAF do pracy z SUT i dodatkowymi konfiguracjami.

Biblioteki podstawowe

W tej warstwie przechowywane są wszystkie biblioteki, które są niezależne od dowolnego SUT. Te podstawowe biblioteki mogą być ponownie wykorzystane w dowolnym typie projektu, który ma ten sam stos programistyczny.

Skalowanie testów automatycznych

Poniższy przykład (Rysunek 3.) pokazuje, w jaki sposób biblioteki podstawowe zapewniają podstawę wielokrotnego użytku dla wielu TAF. W projekcie nr 1 istnieją dwie TAF zbudowane na bazie bibliotek podstawowych, a osobny projekt wykorzystuje już istniejące biblioteki podstawowe do zbudowania ich TAF do testowania aplikacji nr 3. Jeden TAE buduje kolejne TAF na potrzeby projektu nr 1, podczas gdy drugi TAE buduje TAF na potrzeby projektu nr 2.



Rysunek 3. Zastosowanie bibliotek podstawowych do wielu TAF

3.1.4 Kandydat stosuje różne podejścia do automatyzacji przypadków testowych

Istnieje kilka podejść programistycznych, z których zespoły mogą wybierać na potrzeby tworzenia zautomatyzowanych przypadków testowych. Mogą to być interaktywne języki skryptowe lub skompilowane języki programowania. Różne podejścia zapewniają różne korzyści z automatyzacji i można je wykorzystać w różnych okolicznościach. Mimo że wytwarzanie sterowane testami (TDD) i wytwarzanie sterowane zachowaniem (BDD) są metodykami wytwarzania, jeśli są przestrzegane, skutkują one wytwarzaniem zautomatyzowanych przypadków testowych.

Rejestrowanie-odtwarzanie

Rejestrowanie-odtwarzanie jest podejściem, które przechwytuje interakcje z SUT, podczas gdy sekwencja działań jest wykonywana manualnie. Narzędzia te wytwarzają skrypty testowe podczas rejestrowania, a w zależności od używanego narzędzia kod automatyzacji testów może być modyfikowalny. Narzędzia, które nie eksponują kodu, są czasami określane jako automatyczne testy bezkodowe, podczas gdy narzędzia eksponujące kod są określane jako automatyczne testy niskokodowe.

Zalety

- Początkowo łatwa konfiguracja i korzystanie.

Wady

- trudne w utrzymaniu, skalowaniu i ewolucji;

- SUT musi być dostępny podczas rejestrowania przypadku testowego;
- wykonalne tylko dla małego zakresu i SUT, który rzadko się zmienia;
- zarejestrowane wykonanie SUT zależy w dużej mierze od wersji SUT, z której nastąpiło zarejestrowanie;
- rejestrowanie każdego indywidualnego przypadku testowego zamiast ponownego użycia istniejących bloków konstrukcyjnych jest czasochłonne

Skrypty liniowe

Stosowanie skryptów liniowych jest działaniem programistycznym, które nie wymaga niestandardowych bibliotek testowych wykonanych przez TAE i jest używane do pisania i wykonywania skryptów testowych. TAE może wykorzystać dowolne skrypty testowe, które są rejestrowane przez narzędzie do rejestrowania-odtwarzania, które następnie można modyfikować.

Zalety

- łatwa konfiguracja i rozpoczęcie pisania skryptów testowych;
- w porównaniu z rejestrowaniem-odtwarzaniem, skrypty testowe można łatwiej modyfikować.

Wady

- trudne w utrzymaniu, skalowaniu i ewolucji;
- SUT musi być dostępny podczas rejestrowania przypadku testowego;
- wykonalne tylko dla małego zakresu i SUT, który rzadko się zmienia;
- w porównaniu z rejestrowaniem-odtwarzaniem, niezbędna jest pewna wiedza z zakresu programowania.

Skrypty ustrukturalizowane

Biblioteki testowe są wprowadzane z elementami wielokrotnego użytku, krokami testowymi i/lub podróżami użytkowników (ang. *user journeys*). Znajomość programowania jest niezbędna do tworzenia i utrzymywania skryptów testowych w tym podejściu.

Zalety

- łatwe w utrzymaniu, skalowaniu, przenoszeniu, adaptacji i ewolucji;
- logikę biznesową można oddzielić od skryptów testowych.

Wady

- znajomość programowania jest niezbędna;
- początkowa inwestycja w rozwój TAF i zdefiniowanie testaliów jest czasochłonne.

Wytwarzanie sterowane testami (TDD)

Przypadki testowe są definiowane jako część procesu wytwarzania przed wdrożeniem nowej funkcji SUT. Podejście TDD to testowanie, kodowanie i refaktoryzacja lub inaczej znane jako kolor czerwony, zielony i refaktoryzacja. Programista identyfikuje i tworzy jeden przypadek testowy, który zakończy się niepowodzeniem (kolor czerwony). Następnie opracowuje funkcjonalność, dla której przypadek testowy będzie zaliczony (kolor zielony). Kod jest następnie refaktoryzowany, aby go zoptymalizować i przestrzegać zasad czystego kodu. Proces jest kontynuowany wraz z kolejnym testem i kolejnym przyrostem funkcjonalności.

Zalety

- upraszcza tworzenie przypadków testowych na poziomie modułu;
- poprawia jakość kodu i strukturę kodu;
- poprawia testowalność;
- ułatwia osiągnięcie pożądanego pokrycia kodu;
- zmniejsza rozprzestrzenianie się defektów na wyższe poziomy testów;

- poprawia komunikację między programistami, przedstawicielami biznesu i testerami;
- historyjki użytkowników, które nie są weryfikowane za pomocą testów GUI i testów API, mogą szybko osiągnąć kryteria wyjścia w przypadku postępowania zgodnie z TDD.

Wady

- początkowo potrzeba więcej czasu, aby przyzwyczaić się do TDD;
- nieprzestrzeganie TDD może skutkować fałszywym zaufaniem do jakości kodu.

Testowanie sterowane danymi

Testowanie sterowane danymi (ang. *data-driven testing* - DDT) wykorzystuje technikę skryptów ustrukturalizowanych. Skrypty testowe są dostarczane z danymi testowymi (np. pliki .csv, pliki .xlsx i zrzuty bazy danych). Umożliwia to wielokrotne uruchamianie tych samych skryptów testowych z różnymi danymi testowymi.

Zalety

- umożliwia szybką i łatwą ekspansję przypadków testowych poprzez kanały danych;
- koszt dodania nowych testów automatycznych można znacznie obniżyć;
- analitycy testów mogą określić testy automatyczne, wypełniając jeden lub więcej plików danych testowych, które opisują testy; daje to analitykom testów większą swobodę w określaniu testów automatycznych przy mniejszej zależności od technicznych analityków testów.

Wady

- może być konieczne odpowiednie zarządzanie danymi testowymi.

Testowanie sterowane słowami kluczowymi

Przypadki testowania sterowanego słowami kluczowymi (ang. *keyword driven testing* - KDT) to lista lub tabela kroków testowych pochodzących od słów kluczowych i danych testowych, na których działają słowa kluczowe. Słowa kluczowe są definiowane z perspektywy użytkownika. Technika ta jest często opierana na DDT.

Zalety

- analitycy testów i analitycy biznesowi mogą być zaangażowani w tworzenie przypadków testowych dla testów automatycznych, postępując zgodnie z podejściem KDT;
- KDT może być również używane do testów manualnychh niezależnie od testów automatycznych (patrz norma ISO/IEC/IEEE 29119-5 dotycząca testowania sterowanego słowami kluczowymi).

Wady

- wdrażanie i utrzymywanie słów kluczowych jest złożonym zadaniem, które TAE muszą realizować, co może stać się wyzwaniem, gdy zakres będzie się zwiększał;
- generuje ogromny wysiłek w przypadku mniejszych systemów.

Wytwarzanie sterowane zachowaniem

Wytwarzanie sterowane zachowaniem (ang. *behavior driven development* - BDD) wykorzystuje format języka naturalnego (tj. *Given/When/Then*) do formułowania kryteriów akceptacji, które mogą być używane jako zautomatyzowane przypadki testowe i przechowywane w plikach scenariuszowych (ang. *feature files*). Narzędzie BDD może następnie zrozumieć język i wykonać przypadki testowe.

Zalety

- poprawia komunikację między programistami, przedstawicielami biznesu i testerami;
- zautomatyzowane scenariusze BDD działają jako przypadki testowe i zapewniają pokrycie specyfikacji;

- BDD można wykorzystać do produkcji wielu typów testów na różnych poziomach piramidy testów.

Wady

- dodatkowe przypadki testowe, zazwyczaj negatywne warunki testowe i przypadki brzegowe nadal wymagają zdefiniowania przez zespół, zwykle przez analityka testów lub TAE;
- wiele zespołów myli BDD ze sposobem pisania przypadków testowych w języku naturalnym i nie angażuje przedstawicieli biznesu i programistów w całe podejście;
- wdrażanie i utrzymywanie kroków testowych opisanych w języku naturalnym jest złożonym zadaniem dla TAE;
- zbyt skomplikowane kroki testowe sprawią, że debugowanie stanie się trudnym i kosztownym działaniem.

3.1.5 Kandydat stosuje zasady projektowania i wzorce projektowe w automatyzacji testów

Automatyzacja testów to działanie związane z wytwarzaniem oprogramowania. W związku z tym zasady projektowania i wzorce projektowe są tak samo ważne dla TAE, jak dla programisty.

Zasady programowania obiektowego

Istnieją cztery główne zasady programowania obiektowego: enkapsulacja, abstrakcja, dziedziczenie i polimorfizm.

Zasady SOLID

Jest to skrót od „single responsibility, open-closed, Liskov substitution, interface substitution and dependency inversion” (zasada pojedynczej odpowiedzialności, zasada otwarte-zamknięte, zasada podstawienia Liskov, zasada segregacji interfejsów i zasada odwrócenia zależności). Zasady te poprawiają czytelność, utrzymywalność i skalowalność kodu.

Wzorce projektowe

Spośród wielu wzorców projektowych trzy są najważniejsze dla TAE.

Wzorzec fasady ukrywa szczegóły implementacji, aby ujawnić tylko to, co testerzy muszą stworzyć w przypadkach testowych, a wzorzec singleton jest często używany, aby upewnić się, że jest tylko jeden sterownik, który komunikuje się z SUT.

We wzorcu obiektu strony tworzony jest plik klasy, który określa się mianem modelu strony. Ilekroć zmienia się struktura SUT, TAE będzie musiał dokonywać aktualizacji tylko w jednym miejscu (lokalizator wewnątrz modelu strony) zamiast aktualizować lokalizatory w każdym przypadku testowym.

Model przepływu jest rozszerzeniem modelu obiektu strony. Wprowadza dodatkową fasadę nad modelami obiektów stron, która przechowuje wszystkie działania użytkownika, które wchodzą w interakcję z obiektami stron. Wprowadzenie konstrukcji podwójnej fasady sprawia, że wzór modelu przepływu zapewnia lepszą abstrakcję i łatwość utrzymania, ponieważ kroki testowe mogą być ponownie wykorzystane w wielu skryptach testowych.

4 Wdrażanie testów automatycznych – 150 minut (K4)

Słowa kluczowe

ryzyko, uchwyt testowy

Cele nauczania w rozdziale 4:

4.1 Rozwój automatyzacji testów

TAE-4.1.1 (K3) Kandydat stosuje wytyczne, które wspierają skuteczne działania pilotażowe i wdrożeniowe w zakresie automatyzacji testów.

4.2 Ryzyka związane z rozwojem automatyzacji testów

TAE-4.2.1 (K4) Kandydat analizuje ryzyka związane z wdrożeniem i planuje strategię łagodzenia skutków w zakresie automatyzacji testów.

4.3 Utrzymywalność rozwiązania dla testów automatycznych

TAE-4.3.1 (K2) Kandydat wyjaśnia, które czynniki wspierają i wpływają na utrzymywalność rozwiązania dla testów automatycznych.

4.1 Rozwój automatyzacji testów

4.1.1 Kandydat stosuje wytyczne, które wspierają skuteczne działania pilotażowe i wdrożeniowe w zakresie automatyzacji testów

Ważne jest określenie zakresu walidacji na potrzeby projektu pilotażowego automatyzacji testów. Przeprowadzenie projektu pilotażowego nie zajmuje dużo czasu, ale wynik może mieć znaczący wpływ na kierunek, w jakim podąża projekt.

Na podstawie zebranych informacji na temat SUT i wymagań dotyczących projektu należy ocenić następujące elementy w celu ustalenia wytycznych dotyczących optymalizacji działań w zakresie automatyzacji testów:

- języki programowania, które będą używane;
- odpowiednie narzędzia do powszechnej sprzedaży/open source;
- poziomy testów do pokrycia;
- wybrane przypadki testowe;
- podejście do wytwarzania przypadków testowych.

Na podstawie punktów wymienionych powyżej TAE mogą zdefiniować wstępne podejście, które należy stosować. W oparciu o wymagania można utworzyć kilka różnych początkowych prototypów, aby pokazać zalety i wady różnych podejść. Od tego momentu TAE mogą decydować o tym, którą ścieżką podążać dalej.

Określanie ram czasowych jest ważną częścią harmonogramów spotkań i zapewnienia sukcesu projektu pilotażowego. Częstym zaleceniem jest okresowa kontrola postępów projektu pilotażowego w celu zidentyfikowania wszelkich ryzyk i ich złagodzenia.

Podczas programu pilotażowego zaleca się również próbę zintegrowania rozwiązania i już zaimplementowanego kodu z CI/CD. Może to ujawnić problemy na wczesnym etapie, zarówno w SUT, TAS, jak i w ogólnej integracji różnych narzędzi w organizacji.

Wraz ze wzrostem liczby przypadków testowych TAE mogą pomyśleć o zmianie początkowej konfiguracji CI/CD, aby przeprowadzać testy na różne sposoby i w różnym czasie.

Podczas projektu pilotażowego istnieje również potrzeba oceny innych aspektów nietechnicznych, takich jak:

- wiedza i doświadczenie członków zespołu;
- struktura zespołu;
- zasady licencjonowania i organizacji;
- rodzaj planowanych testów i docelowe poziomy testów do pokrycia podczas automatyzacji przypadków testowych.

Po zakończeniu projektu pilotażowego wysiłek powinien zostać oceniony przez TAE i kierowników testów w celu oceny sukcesu lub niepowodzenia i podjęcia odpowiedniej decyzji.

4.2 Ryzyka związane z rozwojem automatyzacji testów

4.2.1 Kandydat analizuje ryzyka związane z wdrażaniem i planuje strategię łagodzenia skutków w zakresie automatyzacji testów

Połączenie TAF z SUT należy rozpatrywać jako część projektu architektonicznego. Następnie można wybrać opakowanie, rejestrowanie testów i narzędzia jarzma testowego.

Podczas wdrażania programu pilotażowego należy rozważyć rozszerzenie i utrzymanie kodu automatyzacji testów. Są to kluczowe czynniki fazy oceny projektu pilotażowego i mogą mieć poważny wpływ na ostateczną decyzję.

W ramach projektu pilotażowego można zidentyfikować różne ryzyka związane z wdrożeniem:

- luki zapory sieciowej;
- zużycie zasobów (np. procesora i pamięci RAM).

Należy przygotować się na ryzyka związane z wdrożeniem, takie jak problemy z zaporą sieciową, zużycie zasobów, połączenie sieciowe i niezawodność. Nie są one ściśle związane z automatyzacją testów, ale TAE muszą zadbać o spełnienie wszystkich warunków, aby zapewnić niezawodne i korzystne bramki jakości w swoim procesie wytwarzania.

Przykładem może być korzystanie z prawdziwych urządzeń do automatyzacji testów mobilnych. Urządzenia mobilne muszą być włączone, mieć wystarczający poziom naładowania akumulatorów do pracy podczas testu, być podłączone do sieci i mieć dostęp do SUT.

Ryzyka techniczne związane z wdrożeniem mogą obejmować:

- tworzenie wydania;
- rejestrowanie;
- strukturyzację testów;
- aktualizowanie.

Tworzenie wydania

Tworzenie wydania należy wziąć pod uwagę, ponieważ kontrola wersji automatyzacji testów jest tak samo ważna, jak w przypadku SUT. Testalia mogą wymagać przesłania do repozytorium, aby udostępnić je całej organizacji, zarówno lokalnie, jak i w chmurze.

Rejestrowanie

Rejestrowanie testów daje większość informacji o wynikach testów. Istnieje kilka poziomów logowania testów i wszystkie z nich są przydatne w automatyzacji testów z różnych powodów:

- Krytyczny błąd: ten poziom służy do rejestrowania zdarzeń awarii, które mogą prowadzić do przerwania wykonywania testu.
- Błąd: ten poziom jest używany, gdy warunek lub interakcja nie powiedzie się, a tym samym nie zostanie zaliczony również przypadek testowy.
- Ostrzeżenie: ten poziom jest używany, gdy wystąpi nieoczekiwany warunek/działanie, ale nie przerwie przepływu przypadku testowego.
- Informacja: ten poziom służy do pokazania podstawowych informacji o przypadku testowym i tym, co dzieje się podczas wykonywania testu.
- Debugowanie: ten poziom służy do przechowywania szczegółowych informacji dotyczących wykonania, które na ogół nie są wymagane do podstawowych logów, ale są przydatne podczas badania niepowodzenia testu.
- Śledzenie: ten poziom jest podobny do debugowania, ale zawiera jeszcze więcej informacji.

Strukturyzacja testów

Najważniejszą częścią TAS jest jarzmo testowe i dołączone do niego uchwyty testowe, elementy, które muszą być dostępne do przeprowadzenia testów. Uchwyty testowe zapewniają swobodę w sterowaniu środowiskiem testowym i danymi testowymi. Warunki wstępne i warunki końcowe można zdefiniować na potrzeby wykonania testu, a przypadki testowe można pogrupować w zestawy testowe na kilka sposobów. Aspekty te są również ważne do oceny podczas projektu pilotażowego. Ponadto uchwyty testowe umożliwiają tworzenie testów automatycznych, które są powtarzalne i atomowe.

Aktualizowanie

Jednym z najczęstszych przypadków ryzyk technicznych są automatyczne aktualizacje jarzm testowych (np. agentów) i zmiany wersji na urządzeniach. Ryzyka te można złagodzić dzięki odpowiednim zasilaczom, odpowiednim połączeniom sieciowym i odpowiednim planom konfiguracji urządzeń.

4.3 Utrzymywalność rozwiązania dla testów automatycznych

4.3.1 Kandydat wyjaśnia, które czynniki wspierają i wpływają na utrzymywalność rozwiązania dla testów automatycznych

Na utrzymywalność duży wpływ mają standardy programowania i wzajemne oczekiwania TAE.

Złotą zasadą jest próba przestrzegania zasad czystego kodu Roberta C. Martina (Robert C. Martin, „Clean Code: A Handbook of Agile Software Craftsmanship", 2008).

W skrócie, zasady czystego kodu podkreślają następujące punkty:

- używanie wspólnej konwencji nazewnictwa klas, metod i zmiennych dla nazw znaczących;
- używanie logicznej i wspólnej struktury projektu;
- unikanie kodowania na stałe (ang. *hardcoding*);
- unikanie zbyt wielu parametrów wejściowych dla metod;
- unikanie długich i złożonych metod;
- używanie logowania;
- stosowanie wzorców projektowych tam, gdzie są one korzystne i wymagane;
- koncentracja na testowalności.

Konwencje nazewnictwa są bardzo pomocne w identyfikacji celu danej zmiennej. Zrozumiałe nazwy zmiennych, takich jak „loginButton", „resetPasswordButton", pomaga TAE zrozumieć, którego komponentu użyć.

Kodowanie na stałe to proces osadzania wartości w oprogramowaniu bez możliwości ich bezpośredniej zmiany. Można tego uniknąć, stosując testowanie oparte na danych, dzięki czemu dane testowe pochodzą ze wspólnego źródła, które można łatwiej utrzymać. Kodowanie na stałe skraca czas programowania, ale nie jest zalecany, ponieważ dane mogą się często zmieniać, co może być czasochłonne w utrzymaniu. Zaleca się również stosowanie stałych dla tych zmiennych, w przypadku których nie oczekuje się częstych zmian. W ten sposób można zmniejszyć źródła i miejsca wymagające utrzymania.

Zalecane jest również korzystanie z wzorców projektowych. Wzorce projektowe, jak opisano w sekcji 3.1.5, umożliwiają wdrożenie ustrukturyzowanego i odpowiednio obsługiwanego kodu automatyzacji testów, o ile wzorce projektowe są prawidłowo wykorzystywane.

Aby zapewnić wysoką jakość kodu testów automatycznych, zaleca się stosowanie analizatorów statycznych. Formaty kodu, takie jak te powszechnie stosowane w IDE, poprawią czytelność kodu testów automatycznych.

Oprócz zasad czystego kodu zaleca się stosowanie uzgodnionej struktury i strategii rozgałęzień w kontroli wersji. Używanie różnych gałęzi do funkcji, wydań i napraw defektów jest pomocne w zrozumieniu zawartości gałęzi.

5 Wdrażanie i strategie wdrażania automatyzacji testów – 90 minut (K3)

Słowa kluczowe

testowanie kontraktu

Cele nauczania w rozdziale 5:

5.1 Integracja z potokami CI/CD

TAE-5.1.1 (K3) Kandydat stosuje automatyzację testów na różnych poziomach testów w potokach.

TAE-5.1.2 (K2) Kandydat wyjaśnia zarządzanie konfiguracją testaliów.

TAE-5.1.3 (K2) Kandydat wyjaśnia zależności automatyzacji testów dla infrastruktury API.

5.1 Integracja z potokami CI/CD

5.1.1 Kandydat stosuje automatyzację testów na różnych poziomach testów w potokach

Jedną z głównych zalet automatyzacji testów jest to, że zaimplementowane testy mogą być uruchamiane bez nadzoru, co czyni je idealnymi kandydatami do uruchamiania w ramach potoków. Można to osiągnąć za pomocą potoków CI/CD lub potoków używanych do regularnego przeprowadzania testów.

Poziomy testów są zwykle zintegrowane w następujący sposób:

- Testy konfiguracji dla TAF/TAS podczas kompilacji można uznać za rodzaj testów modułowych. Testy te są uruchamiane podczas kompilacji projektu automatyzacji testów (TAF/TAS) i sprawdzają, czy wszystkie ścieżki do plików używanych w skryptach testowych są poprawne, czy pliki naprawdę istnieją i znajdują się w określonych ścieżkach.
- Testy modułowe są częścią etapu budowania potoków, ponieważ są wykonywane na poszczególnych komponentach (np. klasach bibliotek i komponentach webowych). Stanowią one bramki jakości dla potoku, a tym samym kluczowy element potoku ciągłej integracji.
- Testy integracji modułów mogą być częścią potoku ciągłej integracji, jeśli są testami komponentów niskiego poziomu SUT. W takich przypadkach te testy oraz testy modułowe są wykonywane razem.
- Testy systemowe często można zintegrować z potokiem ciągłego wdrażania, gdzie działają one jako ostatnia bramka jakości dostarczonego SUT.
- Testy integracji systemów między różnymi modułami systemu są często częścią potoku ciągłego dostarczania w charakterze bramki jakości. Te testy integracji systemów zapewniają, aby oddzielnie opracowane komponenty systemu współpracowały ze sobą.

Wiele nowoczesnych systemów ciągłej integracji rozróżnia fazy budowy i wdrażania potoków ciągłego dostarczania. W takich przypadkach testy modułowe i testy integracji modułów są częścią fazy budowania wersji. Gdy ta pierwsza faza zakończy się powodzeniem (tj. jednoczesne budowanie i testowanie), komponenty/SUT są wdrażane.

W przypadku testowania integracji systemów, testowania systemowego i testowania akceptacyjnego istnieją dwa główne podejścia do ich integracji z takimi potokami:

1. Przypadki testowe są wykonywane w ramach fazy wdrażania po wdrożeniu komponentu. Może to być korzystne, ponieważ w oparciu o wyniki testu wdrożenie może się nie powieść, a także może zostać wycofane. Jednak w tym przypadku, jeśli testy muszą zostać powtórzone, należy przeprowadzić ponowne wdrożenie.
2. Przypadki testowe są wykonywane jako oddzielny potok, wyzwalany przez pomyślne wdrożenie. Może to być korzystne, jeśli oczekuje się, że na każdym wdrożeniu będą działać różne zestawy testowe i różne kody automatyzacji testów. W tym przypadku testy nie działają jak bramka jakości. Wymaga to zatem innych działań, zwykle manualnych, aby wycofać nieudane wdrożenie.
   W tym przypadku można użyć kilku prostych zautomatyzowanych skryptów testowych, takich jak testy wdrożeniowe, aby zapewnić wdrożenie SUT, ale te zautomatyzowane skrypty testowe nie weryfikują przydatności funkcjonalnej w szerokim zakresie.

Potoki mogą być również wykorzystywane do innych celów związanych z automatyzacją testów, takich jak:

- okresowe uruchamianie różnych zestawów testowych: zestaw testów regresji można uruchamiać co noc (tj. regresja nocna), szczególnie w przypadku dłuższych zestawów testowych, dzięki czemu rano zespół otrzyma jasny obraz jakości SUT;

- uruchamianie testów niefunkcjonalnych: jako część potoku ciągłego wdrażania lub osobno, w celu okresowego monitorowania niektórych niefunkcjonalnych cech jakościowych systemu, takich jak wydajność.

5.1.2 Kandydat wyjaśnia zarządzanie konfiguracją testaliów

Zarządzanie konfiguracją jest integralną częścią automatyzacji testów, ponieważ automatyzacja będzie często wykonywana w wielu środowiskach testowych i wersjach SUT.

Zarządzanie konfiguracją w automatyzacji testów obejmuje:

- konfigurację środowiska testowego;
- dane testowe;
- zestawy testowe/przypadki testowe.

Konfiguracja środowiska testowego

Każde środowisko testowe używane w potoku wytwarzania oprogramowania może mieć różne konfiguracje, takie jak różne adresy URL lub poświadczenia. Konfiguracja środowiska testowego jest zwykle przechowywana wraz z testaliami. Mimo to w przypadku automatyzacji testów stosowanej w wielu projektach lub wielu TAF dla tego samego projektu konfiguracja środowiska testowego może być częścią wspólnej biblioteki podstawowej lub we wspólnym repozytorium.

Dane testowe

Dane testowe mogą być również specyficzne dla środowiska testowego lub dla wersji i zestawu funkcji SUT. Podobnie jak w przypadku konfiguracji środowiska testowego, dane testowe są zwykle przechowywane z mniejszymi TAF, ale można również używać systemów zarządzania danymi testowymi.

Zestawy testowe/przypadki testowe

Powszechną praktyką jest tworzenie różnych zestawów testowych w odniesieniu do przypadków testowych, w zależności od ich celu, takich jak testy dymne lub testy regresji. Te pakiety testowe są często wykonywane na oddzielnych poziomach testów, z wykorzystaniem różnych potoków i środowisk testowych.

Każde wydanie SUT określa zestaw funkcji, który obejmuje przypadki testowe i zestawy testowe, które oceniają jakość danego wydania. W tym celu w testaliach dostępne są różne opcje obsługi:

- W odniesieniu do każdej wersji lub środowiska testowego można zdefiniować konfigurację przełączania funkcji (ang. *feature toggle*). Istnieją przypadki testowe i zestawy testowe do testowania każdej funkcji. Przełączanie funkcji może być używane w testaliach do określania, które zestawy testowe należy wykonać w danym wydaniu/środowisku testowym.
- Testalia mogą być również wydane z SUT przy użyciu tej samej wersji. W ten sposób istnieje dokładne dopasowanie między wersją SUT a testaliami, które mogą go przetestować. Takie wydanie jest zwykle wdrażane za pomocą systemu zarządzania konfiguracją z wykorzystaniem tagów lub gałęzi.

5.1.3 Kandydat wyjaśnia zależności automatyzacji testów dla infrastruktury API

Podczas automatyzacji testów API kluczowe jest posiadanie następujących informacji o zależnościach, aby zbudować odpowiednią strategię:

- połączenia API: należy zrozumieć logikę biznesową, która może być testowana automatycznie, a także relacje między interfejsami API;
- dokumentacja API: służy jako punkt odniesienia do automatyzacji testów ze wszystkimi istotnymi informacjami (np. parametrami, nagłówkami i różnymi typami żądań-odpowiedzi obiektów).

Zintegrowane testy automatyczne API mogą być wykonywane przez programistów lub TAE. W przypadku przesunięcia w lewo zaleca się jednak wspieranie i dzielenie testów na różne poziomy.

W sylabusie ISTQB CTFL wspomina się o testowaniu integracji modułów i testowaniu integracji systemów, które można rozszerzyć o najlepszą praktykę zwaną testowaniem kontraktu.

Testowanie kontraktu

Testowanie kontraktu to rodzaj testowania integracyjnego, które sprawdza, czy usługi mogą się ze sobą komunikować oraz czy dane udostępniane między usługami są zgodne z określonym zestawem zasad. Korzystanie z testowania kontraktu zapewnia kompatybilność między dwoma oddzielnymi systemami (np. dwoma mikroserwisami) w celu komunikowania się ze sobą. Wykracza to poza walidację schematu, wymagając od obu stron osiągnięcia konsensusu w sprawie dozwolonego zestawu interakcji przy jednoczesnym zapewnieniu ewolucji w czasie. Rejestruje interakcje, które są wymieniane między poszczególnymi usługami, zapisując je w kontrakcie, który można następnie wykorzystać do sprawdzenia, czy obie strony się do niego stosują. Jedną z głównych zalet tego typu testów jest to, że defekty występujące w podstawowych usługach można znaleźć wcześniej w SDLC, a źródło tych defektów można łatwiej zidentyfikować.

W zorientowanym na konsumenta podejściu do testowania kontraktu konsument wyznacza swoje oczekiwania, określając to, w jaki sposób dostawca powinien reagować na żądania pochodzące od tego konsumenta. W zorientowanym na dostawcę podejściu do testowania kontraktu dostawca tworzy kontrakt, który pokazuje, jak działają jego usługi.

6 Raportowanie i metryki automatyzacji testów – 150 minut (K4)

Słowa kluczowe

dziennik (log) testów, krok testowy, metryka, pomiar, raport o postępie testów

Cele nauczania w rozdziale 6:

6.1 Gromadzenie, analiza i raportowanie danych automatyzacji testów

TAE-6.1.1 (K3) Kandydat stosuje metody gromadzenia danych z rozwiązania dla testów automatycznych i systemu podlegającego testowaniu.

TAE-6.1.2 (K4) Kandydat analizuje dane z rozwiązania dla testów automatycznych i systemu podlegającego testowaniu w celu lepszego zrozumienia wyników testów.

TAE-6.1.3 (K2) Kandydat wyjaśnia, w jaki sposób sporządza się i publikuje raport o postępie testów.

6.1 Gromadzenie, analiza i raportowanie danych automatyzacji testów

6.1.1 Kandydat stosuje metody gromadzenia danych z rozwiązania dla testów automatycznych i systemu podlegającego testowaniu

Dane mogą być zbierane z następujących źródeł:

- dzienniki SUT:
  - sieciowy/mobilny interfejs użytkownika;
  - interfejsy API;
  - aplikacje;
  - serwery WWW;
  - serwery baz danych;
- dzienniki TAF zapewniające ścieżkę audytu;
- dzienniki kompilacji;
- dzienniki wdrażania;
- dzienniki produkcyjne do monitorowania danych w produkcji (patrz sylabus ISTQB CT-PT, podrozdział 2.3):
  - monitorowanie wydajności w produkcji w celu wykonania analizy trendów;
  - dzienniki testów wydajnościowych w środowisku testów wydajnościowych (np. testowanie obciążenia, testowanie przeciążające, testowanie skokowe);
- zrzuty ekranu i nagrania ekranu (natywne dla narzędzia do automatyzacji lub strony trzeciej).

Ponieważ TAS ma zautomatyzowane testalia w swoim rdzeniu, mogą one zostać udoskonalone w celu rejestrowania informacji o użyciu. Udoskonalenia testaliów wprowadzone w podstawowym oprogramowaniu testowym mogą być używane przez wszystkie zautomatyzowane skrypty testowe wyższego poziomu. Na przykład ulepszenie podstawowych testaliów w celu rejestrowania czasu rozpoczęcia i zakończenia wykonywania testu może mieć zastosowanie do wszystkich testów.

Funkcje automatyzacji testów, które obsługują pomiary i generowanie raportów z testów

Języki skryptowe wielu narzędzi testowych obsługują pomiary i raportowanie za pośrednictwem obiektów, które mogą być używane do rejestrowania informacji przed, w trakcie i po wykonaniu poszczególnych testów oraz całych zestawów testowych.

Raportowanie testów z każdego przebiegu testów musi mieć funkcję analizy, aby uwzględnić wyniki poprzednich testów, co umożliwi podkreślenie trendów takich jak zmiany we wskaźniku powodzenia testu.

Automatyzacja testów zwykle wymaga zautomatyzowania zarówno wykonywania testów, jak i weryfikacji testów, przy czym ta ostatnia jest osiągana poprzez porównanie konkretnych elementów rzeczywistych wyników z oczekiwanymi wynikami. Porównanie to najlepiej przeprowadzić za pomocą narzędzia testowego wykorzystującego asercje. Należy wziąć pod uwagę poziom informacji, który jest zgłaszany w wyniku tego porównania. Ważne jest, aby status testu został określony prawidłowo (tj. zaliczony lub niezaliczony). W przypadku niezaliczenia wymagane będą dodatkowe informacje o jego przyczynie (np. zrzuty ekranu).

Różnice między rzeczywistymi wynikami a oczekiwanymi wynikami testu nie zawsze są jasne, a wsparcie narzędziowe może znacznie pomóc w definiowaniu porównań, które ignorują oczekiwane różnice, takie jak daty i godziny, jednocześnie podkreślając wszelkie nieoczekiwane różnice.

Rejestrowanie testów

Dzienniki (logi) testów są źródłem często używanym do analizy potencjalnych defektów w TAS i SUT. W kolejnej sekcji znajdują się przykłady logowania testów, sklasyfikowane według TAS i SUT.

Rejestrowanie TAS

Kontekst określa, czy TAF lub wykonanie testu odpowiada za rejestrowanie informacji, które powinny obejmować, co następuje:

- który przypadek testowy jest obecnie wykonywany, w tym czas rozpoczęcia i zakończenia;
- status wykonania testu, ponieważ podczas gdy awarie można łatwo zidentyfikować w dziennikach testów, TAF powinna również mieć te informacje i zgłaszać je za pośrednictwem tablicy wskaźników. Możliwe statusy wykonania testu to: zaliczony, niezaliczony lub awaria TAS. Status może być czasami niejednoznaczny i ważne jest, aby organizacja zdefiniowała dla nich jasne i spójne definicje. Awaria TAS jest stosowana w sytuacjach, w których defekt nie znajduje się w SUT;
- niskopoziomowe szczegóły dziennika testów (np. rejestrowanie istotnych kroków testów), w tym informacje o czasie;
- dynamiczne informacje o SUT (np. wycieki pamięci), które przypadek testowy był w stanie zidentyfikować za pomocą narzędzi innych firm. Rzeczywiste wyniki i awarie powinny być rejestrowane w zestawie testowym, który był wykonywany w momencie wykrycia awarii;
- w przypadku testowania niezawodności lub testowania przeciążającego, w których wykonuje się wiele cykli testowych, należy zarejestrować licznik, aby łatwo określić, ile razy wykonano przypadki testowe;
- gdy przypadki testowe mają losowe elementy (np. losowe parametry lub losowe kroki testów w testowaniu przejść pomiędzy stanami), losowa liczba/wybory powinny być rejestrowane;
- wszystkie działania wykonywane przez przypadek testowy powinny być rejestrowane w taki sposób, aby dzienniki testów lub ich części mogły zostać odtworzone w celu ponownego uruchomienia testu z tymi samymi krokami testowymi i tym samym czasem. Jest to przydatne do odtworzenia zidentyfikowanej awarii i uzyskania dodatkowych informacji. Informacje o działaniu przypadku testowego mogą być również rejestrowane przez SUT do wykorzystania podczas odtwarzania awarii zidentyfikowanych przez klienta. Jeśli klient uruchamia zestaw testowy, informacje z dziennika testów są przechwytywane, a następnie mogą być odtwarzane przez zespół programistów podczas rozwiązywania problemów z defektem;
- zrzuty ekranu można zapisać podczas wykonywania testu do dalszego wykorzystania podczas analizy przyczyny podstawowej;
- ilekroć zestaw testowy inicjuje awarię, TAS powinno upewnić się, że wszystkie informacje potrzebne do analizy defektu są dostępne/przechowywane, podobnie jak wszelkie informacje dotyczące kontynuacji testów, jeśli ma to zastosowanie. TAS powinno zapisywać wszelkie powiązane zrzuty awarii (ang. *crash dumps*) i ślady stosów (ang. *stack traces*). Ponadto wszelkie dzienniki testów, które można nadpisać (np. bufory cykliczne są często używane do dzienników testów w SUT), powinny być przechowywane w miejscu, w którym będą dostępne do późniejszej analizy;
- użycie koloru może pomóc w rozróżnieniu różnych typów informacji w dzienniku testów (np. defekty w kolorze czerwonym i informacje o postępie w kolorze zielonym).

Rejestrowanie SUT

Korelacja wyników automatyzacji testów z dziennikami SUT pomaga zidentyfikować podstawową przyczynę defektów w SUT i TAS.

- po zidentyfikowaniu defektu w SUT należy zarejestrować wszystkie niezbędne informacje potrzebne do analizy defektu, w tym znaczniki daty i czasu, lokalizację źródła defektu i komunikaty o błędach;

- podczas uruchamiania systemu informacje o konfiguracji powinny być rejestrowane w pliku, składającym się np. z różnych wersji oprogramowania/oprogramowania wbudowanego, konfiguracji SUT i konfiguracji systemu operacyjnego;
- korzystanie z automatyzacji testów pozwala łatwo przeszukiwać dzienniki SUT. Awaria zidentyfikowana w dzienniku testów przez TAS powinna być łatwo zidentyfikowana w dzienniku testów SUT i odwrotnie, z dodatkowymi narzędziami lub bez nich. Synchronizacja różnych dzienników testów ze znacznikiem czasu ułatwia korelację tego, co wystąpiło, gdy awaria została zarejestrowana.

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

Gdy informacje z wykonywania zautomatyzowanych przypadków testowych są wykorzystywane w innych narzędziach do śledzenia i raportowania (np. aktualizacja informacji o identyfikowalności), możliwe jest dostarczenie informacji w formacie odpowiednim dla narzędzi innych firm. Często osiąga się to poprzez istniejącą funkcjonalność narzędzia testowego (np. formaty eksportu do raportowania testów) lub poprzez tworzenie niestandardowych raportów, które są wyprowadzane w formacie zgodnym z innym oprogramowaniem.

Wizualizacja wyników testów

Wyniki testów mogą być wizualizowane za pomocą wykresów. Należy rozważyć użycie kolorowych ikon, takich jak sygnalizacja świetlna, aby wskazać ogólny stan wykonania testu/automatyzacji testu, co pozwoli podejmować decyzje na podstawie raportowanych informacji. Kierownictwo jest szczególnie zainteresowane podsumowaniami wizualnymi, które pozwalają zobaczyć wyniki testów, co pomaga w podejmowaniu decyzji. Jeśli potrzeba więcej informacji, nadal można zagłębić się w szczegóły.

6.1.2 Kandydat analizuje dane z rozwiązania dla testów automatycznych i systemu podlegającego testowaniu w celu lepszego zrozumienia wyników testów

Po wykonaniu testu ważne jest, aby przeanalizować jego wyniki, co pozwoli zidentyfikować możliwe awarie zarówno w SUT, jak i TAS. Do takiej analizy dane zebrane z TAS służą jako pierwotne, a dane zebrane z SUT – jako wtórne.

- Analiza danych środowiska testowego w celu wsparcia prawidłowego rozmiaru automatyzacji testów (np. w chmurze):
  - klastry i zasoby (np. procesor i pamięć RAM);
  - wykonanie testu pojedynczo lub w wielu przeglądarkach (tj. w różnych przeglądarkach);
- porównanie wyników poprzednich testów;
- określenie, jak korzystać z dzienników internetowych do monitorowania korzystania z oprogramowania.

Awarie wykonania testów wymagają analizy, ponieważ istnieją potencjalne problemy:

1. Należy sprawdzić, czy w poprzednich testach wystąpiła ta sama awaria. Może to być znany defekt w SUT lub TAS. TAS może być zbudowane pod kątem rejestrowania historycznych wyników testów przypadków testowych, co dodatkowo pomaga w analizie.
2. Jeśli defekt nie jest znany, należy zidentyfikować przypadek testowy i to, co testuje. Test może być samowyjaśniający lub przypadek testowy można zidentyfikować w systemie zarządzania testami, na podstawie jego identyfikatora zarejestrowanego podczas wykonywania testu.
3. Należy znaleźć krok testowy przypadku testowego, w którym wystąpiła awaria. Rejestruje ją TAS.
4. Konieczna jest analiza informacji z dziennika testów o stanie SUT oraz tego, czy odpowiada oczekiwanym wynikom, za pomocą zrzutów ekranu, API i dzienników sieciowych lub dowolnego dziennika, który pokazuje stan SUT.
5. Jeśli stan SUT nie jest zgodny z oczekiwaniami, należy zarejestrować defekt w systemie zarządzania defektami. Należy dołączyć wszystkie niezbędne informacje o defekcie i dzienniki, które uzasadniają, że jest to defekt.

W przypadku awarii możliwe jest dopasowanie rzeczywistego wyniku i oczekiwanego wyniku SUT. W takim przypadku najprawdopodobniej TAS zawiera defekt, który należy naprawić, lub występuje niewidoczne niedopasowanie.

Inną sytuacją, która może wystąpić, jest sytuacja, w której środowisko testowe nie jest dostępne podczas uruchomienia testu lub jest tylko częściowo dostępne. W takim wypadku wszystkie przypadki testowe mogą zakończyć się niepowodzeniem, albo z tym samym defektem, albo – jeśli części systemu są uszkodzone – z pozornie prawdziwymi awariami. Aby zidentyfikować podstawową przyczynę takich defektów, można przeanalizować dzienniki SUT, które pokażą, czy w czasie wykonywania testu wystąpiły jakiekolwiek przerwy w działaniu środowiska testowego.

Jeśli SUT implementuje dzienniki audytu dla interakcji użytkowników (tj. sesji interfejsu użytkownika lub wywołań API), pomaga analizować wyniki testów. Zwykle do interakcji dodawany jest unikalny identyfikator o tym samym identyfikatorze dla każdego kolejnego wywołania i integracji w systemie. W ten sposób, znając unikalny identyfikator wywołania/interakcji, można zaobserwować i prześledzić zachowanie systemu.

Ten unikalny identyfikator jest zwykle nazywany identyfikatorem korelacji lub identyfikatorem śledzenia. Może on zostać zarejestrowany przez TAS, aby pomóc w analizie wyników testów.

6.1.3 Kandydat wyjaśnia, w jaki sposób sporządza się i publikuje raport o postępie testów

Dzienniki testów zawierają szczegółowe informacje o krokach testowych, działaniach, które należy podjąć, i oczekiwanych odpowiedziach przypadku testowego i/lub zestawu testowego. Mimo to same dzienniki testów nie mogą zapewnić dobrego przeglądu ogólnych wyników testów. W tym celu konieczne jest dysponowanie funkcją raportowania testów. Po wykonaniu zestawu testowego należy utworzyć i opublikować zwięzły raport o postępie testów. Można w tym celu użyć generatora raportów.

Treść raportu o postępie testów

Raport o postępie testów musi zawierać wyniki testów, informacje o SUT i dokumentację środowiska testowego, w którym testy zostały przeprowadzone, w formacie odpowiednim dla każdego z interesariuszy.

Konieczna jest wiedza o tym, które testy zakończyły się niepowodzeniem i jakie są jego przyczyny. W celu łatwiejszego rozwiązywania problemów ważne jest, aby znać historię wykonywania testów i kto ją zgłosił (tj. zazwyczaj osoba, która ją utworzyła lub ostatnio zaktualizowała). Osoba odpowiedzialna musi zbadać przyczynę niepowodzenia, zgłosić związany z nim defekt, podjąć dalsze działania związane z poprawką oraz przetestować, czy poprawka została prawidłowo wdrożona.

Raportowanie testów służy również do diagnozowania wszelkich niepowodzeń komponentów TAF.

Publikowanie raportów z testów

Raport z testów powinien zostać opublikowany wszystkim odpowiednim interesariuszom. Można go przesłać za pomocą strony internetowej, chmury lub lokalnie, wysłać przez listę mailingową lub przesłać do innego narzędzia, takiego jak narzędzie do zarządzania testami. Pomaga to zapewnić, aby raporty były sprawdzane i analizowane, jeśli osoby są zapisane do listy i otrzymują wiadomości pocztą elektroniczną, lub za pośrednictwem wiadomości na czacie opublikowanych przez czatbota.

Opcją jest zidentyfikowanie problematycznych części SUT i prowadzenie historii raportów z testów, aby można było gromadzić statystyki dotyczące przypadków testowych lub zestawów testowych z częstymi regresjami w celu analizy trendów.

Interesariusze, którym należy przesłać raport:

- Zarząd:
  - zazwyczaj: architekt rozwiązań lub projektant architektury całościowej (ang. *enterprise architect*), kierownik projektu/dostaw, kierownik programu, kierownik testów lub dyrektor testów;
- Interesariusze operacyjni:

- zazwyczaj: właściciel/kierownik produktu, przedstawiciel firmy lub analityk biznesowy;

- Interesariusze techniczni:
  - zazwyczaj: lider zespołu, scrum master, administrator sieci, programista/administrator bazy danych, lider testów, TAE, tester lub programista.

Raporty z testów mogą różnić się treścią lub szczegółowością w zależności od odbiorców. Podczas gdy interesariusze techniczni mogą być bardziej zainteresowani szczegółami niższego poziomu, kierownictwo skupi się na trendach, takich jak liczba przypadków testowych dodanych od ostatniego testu, zmiany we współczynniku zaliczenia/niezaliczenia oraz niezawodność TAS i SUT. Interesariusze operacyjni zazwyczaj kładą większy nacisk na metryki związane z wykorzystaniem produktów.

Tworzenie tablic wskaźników

Nowoczesne narzędzia do raportowania zapewniają kilka opcji raportowania za pomocą tablic wskaźników, kolorowych wykresów, szczegółowych zbiorów dzienników i zautomatyzowanej analizy dzienników testów. Na rynku dostępnych jest wiele narzędzi do wyboru.

Narzędzia te obsługują agregację danych ze źródeł, takich jak dzienniki testów wykonania potoku, narzędzia do zarządzania projektami i repozytoria kodu. Wizualizacja danych dostarczanych przez te narzędzia pomaga interesariuszom dostrzegać trendy i odpowiednio podejmować decyzje. Trendy te mogą obejmować klastry defektów, wzrost/spadek propagacji defektów do niektórych środowisk testowych, pogorszenie wydajności SUT i niezawodność kompilacji.

Analiza dzienników testów przez sztuczną inteligencję/uczenie maszynowe

W ostatnich latach niektóre narzędzia do automatyzacji testów zawierają algorytmy uczenia maszynowego (ML) lub są na nich oparte. Zautomatyzowana analiza dużych ilości danych w dziennikach testów pomaga TAE skrócić czas spędzany na wyszukiwaniu uszkodzonych lokalizatorów, analizowaniu przyczyny niepowodzeń testów (tzn. czy jest to defekt w SUT, czy w TAS) i grupowaniu typowych defektów na potrzeby raportowania testów (patrz sylabus ISTQB CT-AI).

7 Weryfikacja rozwiązania dla testów automatycznych – 135 minut (K3)

Słowa kluczowe

analiza statyczna

Cele nauczania w rozdziale 7:

7.1 Weryfikacja infrastruktury testów automatycznych

TAE-7.1.1 (K3) Kandydat planuje weryfikację środowiska automatyzacji testów, w tym konfigurację narzędzia testowego.

TAE-7.1.2 (K2) Kandydat wyjaśnia prawidłowe zachowanie danego skryptu testowego dla testów automatycznych i/lub zestawu testowego.

TAE-7.1.3 (K2) Kandydat identyfikuje miejsca, w których testy automatyczne dają nieoczekiwane wyniki.

TAE-7.1.4 (K2) Kandydat wyjaśnia, w jaki sposób analiza statyczna może pomóc w testowaniu jakości kodu automatyzacji.

7.1 Weryfikacja infrastruktury testów automatycznych

7.1.1 Kandydat planuje weryfikację środowiska automatyzacji testów, w tym konfigurację narzędzia testowego

To, czy środowisko automatyzacji testów i wszystkie inne komponenty TAS działają zgodnie z oczekiwaniami, nadal wymaga weryfikacji. Kontrole te są wykonywane np. przed rozpoczęciem automatyzacji testów. Można podjąć kroki w celu weryfikacji komponentów środowiska automatyzacji testów. Każdy z nich opisano szczegółowo poniżej.

Instalacja, przygotowanie, konfiguracja i dostosowanie narzędzia testowego

TAS składa się z wielu komponentów. Poleganie na każdym z nich pozwala zapewnić niezawodną i powtarzalną wydajność. U podstaw TAS leżą komponenty wykonywalne, odpowiadające im biblioteki funkcji oraz pomocnicze pliki danych i konfiguracji. Proces konfiguracji TAS może obejmować zakres od użycia zautomatyzowanych skryptów instalacyjnych do ręcznego umieszczania plików w odpowiednich folderach. Narzędzia testowe, podobnie jak systemy operacyjne i inne oprogramowanie, regularnie otrzymują Service Pack lub mogą mieć opcjonalne lub wymagane dodatki w celu zapewnienia zgodności z danym środowiskiem SUT.

Zautomatyzowana instalacja lub kopiowanie z repozytorium ma swoje zalety. Może zagwarantować, że testy na różnych SUT zostały przeprowadzone z tą samą wersją TAS i tą samą konfiguracją TAS, jeśli jest to właściwe. Aktualizacje TAS można wprowadzać za pośrednictwem repozytorium. Wykorzystanie repozytorium i proces aktualizacji do nowej wersji TAS powinny być takie same jak w przypadku standardowych narzędzi programistycznych.

Powtarzalność ustanowienia/rozmontowania środowiska testowego

TAS zostanie wdrożone na różnych systemach, serwerach i w celu obsługi potoków CI/CD. Aby zapewnić prawidłowe działanie TAS w każdym środowisku testowym, konieczne jest systematyczne podejście do ładowania i usuwania TAS z dowolnego środowiska testowego. Jest to osiągane z powodzeniem, gdy budowa i przebudowa TAS nie zapewnia dostrzegalnej różnicy w sposobie działania w wielu środowiskach testowych. Zarządzanie konfiguracją komponentów TAS zapewnia niezawodne tworzenie konfiguracji. Po osiągnięciu tego celu udokumentowanie różnych komponentów, które składają się na TAS, dostarczy niezbędnej wiedzy na temat tego, na jakie aspekty TAS może mieć wpływ lub które wymagają zmiany w przypadku zmiany środowiska SUT.

Łączność z systemami wewnętrznymi i zewnętrznymi/interfejsami

Po zainstalowaniu TAS w danym środowisku SUT, a przed rozpoczęciem korzystania z SUT, należy zastosować zestaw kontroli lub warunków wstępnych, aby upewnić się, że dostępna jest łączność z systemami wewnętrznymi, systemami zewnętrznymi i interfejsami. Dobrą praktyką jest np. logowanie się na serwerach, uruchamianie narzędzi do automatyzacji testów, sprawdzanie, czy narzędzia do automatyzacji testów mogą uzyskać dostęp do SUT, manualne sprawdzanie ustawień konfiguracji i upewnianie się, że uprawnienia są ustawione prawidłowo do rejestrowania testów i raportowania testów między systemami. Ustanowienie warunków wstępnych automatyzacji testów jest niezbędne do upewnienia się, że TAS zostało prawidłowo zainstalowane i skonfigurowane.

Testowanie komponentów TAF

Podobnie jak w przypadku każdego projektu wytwarzania oprogramowania, komponenty TAF muszą zostać indywidualnie przetestowane i zweryfikowane. Może to obejmować testy funkcjonalne i niefunkcjonalne (np. wydajność i zużycie zasobów). Np. komponenty, które zapewniają weryfikację obiektu w systemach GUI, muszą być testowane pod kątem szerokiego zakresu klas obiektów, aby ustalić, czy weryfikacja obiektu działa prawidłowo. Podobnie dzienniki testów i raporty z testów powinny generować dokładne informacje dotyczące statusu automatyzacji testów i zachowania SUT. Przykłady testów niefunkcjonalnych mogą obejmować zrozumienie pogorszenia wydajności TAF, zużycia

zasobów systemowych, które mogą wskazywać na defekty, takie jak wycieki pamięci, oraz brak współdziałania komponentów w TAF i/lub poza nią.

7.1.2 Kandydat wyjaśnia prawidłowe zachowanie danego skryptu testowego dla testów automatycznych i/lub zestawu testów

Zautomatyzowane zestawy testowe muszą być testowane pod kątem kompletności, spójności i prawidłowego zachowania. Można zastosować różne rodzaje kontroli weryfikacyjnych, aby upewnić się, że zautomatyzowany zestaw testowy jest dostępny w dowolnym momencie, lub aby ustalić, czy nadaje się do użytku.

Można podjąć kroki w celu weryfikacji zautomatyzowanego zestawu testowego. Są to:

- sprawdzenie zawartości zestawu testowego;
- weryfikacja nowych testów, które koncentrują się na nowych funkcjach TAF;
- rozważenie powtarzalności testów;
- rozważenie inwazyjności zautomatyzowanych narzędzi testowych.

Każdy z nich opisano szczegółowo poniżej.

Sprawdzenie zawartości zestawu testowego

Należy sprawdzić kompletność (np. czy wszystkie przypadki testowe mają oczekiwane wyniki, a dane testowe są obecne) oraz poprawność wersji TAF i SUT.

Weryfikacja nowych testów, które koncentrują się na nowych funkcjach TAF

Przy pierwszym użyciu nowej funkcji TAF w przypadkach testowych należy ją dokładnie zweryfikować i monitorować, aby upewnić się, że działa prawidłowo.

Rozważenie powtarzalności testów

Podczas powtarzania testów wyniki testów powinny być zawsze takie same. Posiadanie przypadków testowych w zestawie testowym, które nie dają wiarygodnego wyniku testu (np. z powodu zjawiska wyścigów – ang. *race condition*), powinno zostać przeniesione z aktywnego zautomatyzowanego zestawu testowego i przeanalizowane oddzielnie w celu znalezienia przyczyny podstawowej. W przeciwnym razie testy będą wymagały poświęcenia znacznej ilości czasu w celu przeanalizowania niepowodzenia.

Rozważenie inwazyjności zautomatyzowanych narzędzi testowych

TAS będzie często ściśle powiązane z SUT. Dzieje się tak z założenia, aby zapewnić lepszą kompatybilność w odniesieniu do poziomu interakcji. Ta ścisła integracja może jednak prowadzić również do niekorzystnych skutków. Np. gdy TAS znajduje się w środowisku SUT, funkcjonalność SUT może różnić się od sytuacji, w której testy są przeprowadzane manualnie, co może również wpływać na wyniki.

Wysoki poziom inwazyjności może wykazać niepowodzenia podczas testów, które nie są widoczne w systemie produkcyjnym. Jeśli spowoduje to niepowodzenia testów automatycznych, zaufanie do TAS może drastycznie spaść. Programiści mogą wymagać, aby awarie zidentyfikowane przez testy automatyczne były powielane manualnie, jeśli to możliwe, aby pomóc w analizie.

7.1.3 Kandydat identyfikuje miejsca, w których testy automatyczne dają nieoczekiwane wyniki

Gdy skrypt testowy nie zostanie zaliczony lub niespodziewane zosatnie zaliczony, należy przeprowadzić analizę przyczyny podstawowej. Obejmuje to sprawdzanie dzienników testów, danych dotyczących wydajności i ustanowienia oraz rozmontowania skryptu testowego.

Pomocne jest również wykonanie kilku odizolowanych testów. Nieregularne awarie są trudniejsze do przeanalizowania. Defekt może występować w przypadku testowym, SUT, TAF, sprzęcie lub sieci. Zasoby systemu monitorowania mogą dostarczyć wskazówek dotyczących przyczyny podstawowej. Analiza pliku dziennika testów przypadku testowego, SUT i TAF może pomóc w identyfikacji

podstawowej przyczyny defektu. Może być również konieczne debugowanie. Pomoc w identyfikacji podstawowej przyczyny może wymagać wsparcia analityka testów, analityka biznesowego, programisty lub inżyniera systemu.

Należy zweryfikować, czy wszystkie asercje są na miejscu. Brak asercji może skutkować niejednoznacznymi wynikami testów.

7.1.4 Kandydat wyjaśnia, w jaki sposób analiza statyczna może pomóc w testowaniu jakości kodu automatyzacji

Analiza statyczna kodu może pomóc w znalezieniu podatności i defektów w kodzie programu. Może to obejmować SUT lub TAF.

Automatyczne skanowanie może sprawdzać kod w celu złagodzenia ryzyk. Zapewnia to przegląd SUT pod kątem defektów i zapewnia przestrzeganie i stosowanie standardów jakości kodu. Można to również uznać za proaktywną technikę wykrywania defektów, która odgrywa ważną rolę w implementacjach DevSecOps (tj. DevOps z naciskiem na zabezpieczenia). Skanowanie to występuje na wczesnym etapie SDLC za pośrednictwem potoków, aby zapewnić zespołom programistycznym natychmiastowe informacje zwrotne.

Wyniki dotyczące defektów są zwykle klasyfikowane jako krytyczne, o wysokiej, średniej lub niskiej krytyczności, dzięki czemu zespoły programistyczne mają możliwość priorytetyzacji defektów, które zdecydują się naprawić. Niektóre narzędzia do analizy statycznej mają również możliwość dostarczania sugerowanych poprawek kodu w celu usunięcia znalezionych defektów. Daje to zespołom wytwórczym kopię błędnych linii kodu i oferuje ewentualny środek zaradczy do wdrożenia przez programistów. Ponadto narzędzia te wspierają TAE, mierząc jakość, sugerując obszary, w których należy skomentować kod, poprawiając projekt kodu w celu zoptymalizowanej obsługi zasobów (np. wykorzystując bloki try/catch i lepsze struktury pętli) oraz usuwając słabe wywołania biblioteki.

Narzędzia do automatyzacji testów wykorzystują języki programowania, dlatego istnieje ryzyko, że do SDLC można wprowadzić nieodpowiedni kod automatyzacji testów. Przykład: powszechną praktyką podczas automatyzacji testów jest posiadanie nazwy użytkownika i hasła. Możliwe, że TAE może nieprawidłowo umieścić hasło w postaci zwykłego tekstu w jednym lub wielu skryptach testowych.

Narzędzia do analizy statycznej mogą korzystnie wpłynąć na kod automatyzacji testów. Mogą one służyć do analizy kodu automatyzacji testów pod kątem naruszeń zabezpieczeń, takich jak hasło umieszczone w formie zwykłego tekstu w kodzie. Narzędzia do analizy statycznej obsługują wiele języków programowania, w tym te używane przez oprogramowanie do automatyzacji testów. W związku z tym konieczne jest, aby TAE rozszerzył najlepsze praktyki skanowania kodu o kod automatyzacji testów. Mimo że kod automatyzacji testów niekoniecznie jest wdrażany wraz z samym oprogramowaniem, istnieją wyraźnie potencjalne podatności na zagrożenia zabezpieczeń, jeśli hasło zostało wykryte w towarzyszącym skrypcie automatyzacji testów (patrz sylabus ISTQB CT-SEC).

8 Ciągłe doskonalenie – 210 minut (K4)

Słowa kluczowe

histogram testowy, walidacja schematu

Cele nauczania w rozdziale 8:

8.1 Możliwości ciągłego doskonalenia testów automatycznych

TAE-8.1.1 (K3) Kandydat poznaje możliwości poprawy przypadków testowych poprzez gromadzenie i analizę danych.

TAE-8.1.2 (K4) Kandydat analizuje aspekty techniczne wdrożonego rozwiązania dla testów automatycznych i przedstawia zalecenia dotyczące udoskonaleń.

TAE-8.1.3 (K3) Kandydat przeprowadza restrukturyzację testaliów do testów automatycznych w celu ich dostosowania do aktualizacji SUT.

TAE-8.1.4 (K2) Kandydat podsumowuje możliwości korzystania z narzędzi do automatyzacji testów.

8.1 Możliwości ciągłego doskonalenia testów automatycznych

8.1.1 Kandydat poznaje możliwości poprawy przypadków testowych poprzez gromadzenie i analizę danych

Gromadzenie i analizę danych można poprawić, rozważając różne rodzaje danych za pomocą podejść opisanych poniżej.

Histogram testowy

Wizualny raport danych testowych przedstawiony jako histogram testowy zapewnia potencjalne obszary udoskonaleń dotyczące trendów danych przypadków testowych. TAE mogą decydować o możliwych obszarach udoskonaleń, ponieważ wiele narzędzi CI/CD i raportowania testów ma możliwość wyświetlania różnych wyników testów i ich odpowiednich danych testowych (np. dzienników wyjątków, komunikatów błędów i zrzutów ekranu). Histogram testowy umożliwia również TAE identyfikację i wybór tych przypadków testowych, które są wrażliwe, i przebudowanie ich z dodatkowymi udoskonaleniami lub poprzez ponowne przemyślenie rzeczywistej implementacji.

Sztuczna inteligencja

Inną najnowszą szansą jest wykorzystanie sztucznej inteligencji (AI) do wspierania testów i automatyzacji testów. Na przykład w przypadkach testowych UI dane obejmują również wartości lokalizatora UI, które można traktować jako dane wejściowe. Najnowsze przełomowe narzędzia mają możliwość wykrywania, czy dany lokalizator został zmieniony z używanego. Na podstawie ML i rozpoznawania obrazu mogą zidentyfikować nowe selektory i użyć algorytmu samoleczenia (ang. *self-healing algorithm*), aby naprawić przypadek testowy i uwzględnić zmienione lokalizatory w raporcie z testu. Może to przyspieszyć kolejne kroki, takie jak zmiany kontroli wersji i utrzymanie kodu.

Walidacja schematu

Walidacja schematu może być stosowana w analizie danych API (np. właściwości pochodzących z docelowych punktów końcowych) i analizie bazy danych (np. zasady walidacji pól oprogramowania, takich jak dozwolone typy danych i zakresy wartości).

Dzięki walidacji schematu TAS jest w stanie sprawdzić, czy odpowiedź jest zgodna ze specyfikacją biznesową. Ten rodzaj sprawdzenia może być użyty do ustalenia, czy obowiązkowe elementy odpowiedzi są obecne w odpowiedzi usługi i czy ich typ obiektu odpowiada zdefiniowanemu w schemacie. W przypadku naruszenia schematu rozwiązanie zwraca rzeczywistą walidację, która pomaga TAE zidentyfikować podstawową przyczynę problemu.

Przykład: API ma sześć obowiązkowych elementów odpowiedzi, które muszą być ciągami w odpowiedzi. Dzięki narzędziom do walidacji schematu nie jest wymagane zapisywanie indywidualnych asercji w celu sprawdzenia, czy typy te są ciągami, a ich wartości nie są puste. Walidacja schematu będzie obsługiwać te kontrole, dzięki czemu zaimplementowany kod automatyzacji testów będzie znacznie krótszy oraz zwiększy się skuteczność wykrywania defektów w usłudze backendowej.

8.1.2 Kandydat analizuje aspekty techniczne wdrożonego rozwiązania dla testów automatycznych i przedstawia zalecenia dotyczące udoskonaleń

Oprócz bieżących zadań w zakresie utrzymania, które są niezbędne do utrzymania synchronizacji TAS z SUT, istnieje wiele możliwości udoskonalenia TAS. Udoskonalenia te można wprowadzić w celu osiągnięcia szeregu korzyści, w tym większej wydajności (np. dalszego ograniczenia manualnych interwencji), lepszej łatwości użytkowania, dodatkowych możliwości i ulepszonego wsparcia dla testów. Na decyzję o tym, jak udoskonalić TAS, wpływa to, jakie cechy dodają projektowi największej wartości.

Konkretne obszary TAS, których udoskonalenie można rozważyć, obejmują skrypty, wykonywanie testów, weryfikację, TAA, TAF, ustanowienie i rozmontowanie, dokumentację, funkcje TAS oraz aktualizacje i modernizacje TAS. Zostały one opisane szczegółowo poniżej.

Skrypty

Istnieją różne techniki tworzenia skryptów: od skryptów liniowych, przez podejście do testowania sterowane danymi, aż po bardziej wyrafinowane podejście oparte na słowach kluczowych, jak opisano w sekcji 3.1.4. Może być wskazane zmodernizowanie obecnej techniki skryptowej TAS w odniesieniu do wszystkich nowych testów automatycznych. Technikę można zmodernizować do wszystkich istniejących testów automatycznych lub przynajmniej tych, które wymagają największego wysiłku w zakresie utrzymania.

Kolejny obszar doskonalenia TAS na potrzeby skryptów testowych może koncentrować się na ich implementacji. Na przykład:

- Dokonanie oceny tego, czy skrypt testowy/przypadek testowy/krok testowy nakładają się na siebie, aby skonsolidować testy automatyczne. Przypadki testowe zawierające podobne sekwencje działań nie powinny implementować tych kroków testowych wielokrotnie. Te kroki testowe należy przekształcić w funkcję i dodać do biblioteki, aby można je było ponownie wykorzystać. Te funkcje biblioteki mogą być następnie używane przez różne przypadki testowe. Poprawia to utrzymywalność testaliów. Gdy kroki testowe nie są identyczne, ale podobne, konieczna może być parametryzacja. Uwaga: jest to typowe podejście do testowania opartego na słowach kluczowych.
- Ustanowienie procesu usuwania awarii w odniesieniu do TAS i SUT. Gdy podczas wykonywania zestawu testowego wystąpi awaria, TAS powinno być w stanie odzyskać sprawność po tym stanie, aby kontynuować następny możliwy test. Gdy wystąpi awaria w SUT, TAS musi wykonać niezbędne działania naprawcze na SUT (np. ponowne uruchomienie SUT), jeśli jest to wykonalne i praktyczne.
- Ocena mechanizmów oczekiwania, aby upewnić się, że używany jest najlepszy typ. Istnieją trzy powszechne mechanizmy oczekiwania:
  - Zakodowane na stałe (ang. *hard coded*) oczekiwanie (tj. oczekiwanie przez określoną liczbę milisekund), które może być podstawową przyczyną wielu defektów automatyzacji testów, biorąc pod uwagę nieprzewidywalność czasów reakcji oprogramowania;
  - dynamiczne oczekiwanie poprzez odpytywanie (np. sprawdzanie, czy nastąpiła określona zmiana stanu lub działanie), co jest znacznie bardziej elastyczne i efektywne:
    - TAS czeka tylko na potrzebną ilość czasu, a żaden czas testowy nie jest marnowany;
    - gdy proces potrwa dłużej niż oczekiwano, odpytanie poczeka, aż warunek zostanie spełniony. Należy pamiętać, aby dołączyć mechanizm limitu czasu – w przeciwnym razie test może czekać w nieskończoność, jeśli wystąpi defekt;
  - jeszcze lepszym sposobem jest subskrybowanie mechanizmu zdarzeń SUT. Jest to znacznie bardziej niezawodne niż dwie pozostałe opcje, ale testowy język skryptowy musi obsługiwać subskrypcję zdarzeń, a SUT musi udostępniać te zdarzenia do TAS. Wymagany jest również mechanizm limitu czasu – w przeciwnym razie test może czekać w nieskończoność, jeśli wystąpi defekt.

Wykonywanie testów

Gdy zestaw automatycznych testów regresji nie zostaje zakończony, ponieważ wykonanie testu trwa zbyt długo, może być konieczne jednoczesne przetestowanie na różnych środowiskach testowych, jeśli jest to możliwe. Gdy do testowania używane są drogie systemy, może istnieć ograniczenie, że wszystkie testy muszą być wykonywane na jednym systemie docelowym. Może być konieczne podzielenie zestawu testów regresji na wiele części, z których każda będzie wykonywana w określonym czasie (np. w ciągu jednej nocy). Dalsza analiza pokrycia testami automatycznymi może ujawnić duplikację. Usunięcie duplikacji może skrócić czas wykonywania testów i może przynieść dalsze korzyści.

W przypadku CI/CD dobrą praktyką jest równoległe uruchamianie zadań wsadowych w celu optymalizacji czasu wykonywania testów. Ponadto dobrą praktyką jest planowanie automatycznych zadań wsadowych, aby uruchamiać różne potoki o określonej porze, np. każdego ranka, co pozwoli ograniczyć manualne interakcje i przyspieszy proces wytwarzania.

Weryfikacja

Przed utworzeniem nowych funkcji weryfikacji należy przyjąć zestaw standardowych metod weryfikacji do użytku we wszystkich testach automatycznych. Pozwoli to uniknąć ponownego wdrażania działań weryfikacyjnych w wielu testach. Gdy metody weryfikacji nie są identyczne, ale podobne, zastosowanie parametryzacji pomoże w użyciu funkcji w wielu typach obiektów.

TAA

Może być konieczna zmiana TAA w celu poprawy testowalności SUT. Zmiany te mogą być wprowadzane w architekturze SUT i/lub w TAA TAS. Może to zapewnić znaczną poprawę automatyzacji testów, ale może wymagać znacznych zmian i inwestycji w SUT/TAS. Jeśli np. SUT ma zostać zmieniony w celu zapewnienia interfejsów API do testowania, należy również odpowiednio zrefaktoryzować TAS. Dodanie tego rodzaju funkcji później w SDLC może być dość kosztowne; znacznie lepiej jest pomyśleć o tym na początku automatyzacji testów i we wczesnych fazach SDLC SUT.

TAF

Często pojawiają się nowe wersje bibliotek podstawowych używanych w TAF. Czasami są to duże aktualizacje, a najnowsza wersja nie może być natychmiast uwzględniona na liście zależności TAF, ponieważ spowodowałoby to awarię testów u wielu zespołów, które z nich korzystają. W związku z tym lepiej najpierw przeprowadzić projekt pilotażowy i analizę wpływu. Następnie można utworzyć plan wdrożenia. Albo wszystkie zespoły przyjmują nową wersję bibliotek podstawowych w tym samym czasie, aktualizując zależność w pliku kompilacji warstwy bibliotek podstawowych, albo każdy zespół indywidualnie decyduje, kiedy zaktualizować ją w swojej warstwie logiki biznesowej. Ostatecznie, gdy wszystkie zespoły będą gotowe do zaakceptowania nowej wersji bibliotek podstawowych, zależności można zaktualizować w warstwie bibliotek podstawowych (patrz sekcja 3.1.3).

Ustanowienie i rozmontowanie

Działania i konfiguracje, które są powtarzane przed lub po każdym skrypcie testowym lub zestawie testowym, należy przenieść do metod ustanowienia lub rozmontowania. W ten sposób wszelkie zmiany, które mają wpływ na kod, mogą być aktualizowane w jednym miejscu, co skutkuje zmniejszeniem nakładów związanych z utrzymaniem. Na przykład wywołania usług internetowych mogą być używane do spełniania warunków wstępnych lub warunków końcowych testów interfejsu użytkownika (np. rejestracja użytkownika, czyszczenie użytkownika i konfiguracja profilu).

Dokumentacja

Obejmuje wszystkie formy dokumentacji: od dokumentacji automatyzacji testów (np. co robi kod automatyzacji testów i jak powinien być używany), po dokumentację użytkownika dla TAS oraz raporty z testów i dzienniki testów tworzone przez TAS.

Funkcje TAS

Do TAS należy dodać funkcje i cechy, takie jak szczegółowe raportowanie testów, dzienniki testów i integracja z innymi systemami. Należy dodawać tylko te nowe funkcje, które będą używane. Dodawanie nieużywanych funkcji tylko zwiększa złożoność oraz zmniejsza niezawodność i utrzymywalność.

Aktualizacje i modernizacje TAF

Poprzez aktualizację lub modernizację do nowych wersji TAF mogą stać się dostępne nowe funkcje, które mogą być używane przez przypadki testowe lub mogą zostać usunięte awarie. Istnieje ryzyko, że aktualizacja TAF poprzez modernizację istniejących narzędzi testowych lub wprowadzenie nowych narzędzi może mieć negatywny wpływ na istniejące przypadki testowe. Należy przetestować najnowszą

wersję narzędzia testowego, uruchamiając przykładowe testy przed wprowadzeniem nowej wersji narzędzia testowego. Testy próbne powinny być reprezentatywne dla zautomatyzowanych testów różnych SUT, różnych typów testów oraz, w stosownych przypadkach, różnych środowisk testowych.

8.1.3 Kandydat przeprowadza restrukturyzację testaliów do testów automatycznych w celu ich dostosowania do aktualizacji SUT

Zastosowanie danego zestawu zmian w istniejącym SUT będzie wymagało aktualizacji TAS, w tym TAF i bibliotek komponentów. Każda zmiana, niezależnie od tego, jak trywialna, może mieć szeroki negatywny wpływ na niezawodność i wydajność TAS.

Identyfikacja zmian w komponentach środowiska testowego

Należy ocenić, jakie zmiany i udoskonalenia wymagają wprowadzenia. Czy wymagają one zmian w testaliach, dostosowanych bibliotekach funkcji lub systemie operacyjnym? Każdy z tych elementów ma wpływ na zachowanie TAS. Celem nadrzędnym jest zapewnienie, aby testy automatyczne nadal przebiegały w sposób efektywny. Zmiany powinny być wprowadzane stopniowo, z nastawieniem na pierwszą wersję produktu (MVP), tak aby wpływ na TAS można było zmierzyć za pomocą ograniczonej serii skryptów testowych. Po stwierdzeniu, że nie ma żadnych skutków ubocznych, zmiany mogą zostać w pełni wdrożone. Pełne uruchomienie regresji jest ostatnim krokiem w kierunku sprawdzenia, czy zmiana nie wpłynęła negatywnie na zautomatyzowane skrypty testowe. Podczas wykonywania tych skryptów testu regresji mogą wystąpić awarie. Identyfikacja podstawowej przyczyny tych awarii (np. poprzez raportowanie testów, dzienniki testów i analizę danych testowych) zapewni środki do upewnienia się, że nie wynikają one z czynności doskonalenia testów automatycznych.

Zwiększenie wydajności i efektywności podstawowych bibliotek funkcji TAS

W miarę dojrzewania TAS odkrywane są nowe sposoby wydajniejszego wykonywania zadań. Te nowe techniki (np. optymalizacja kodu w funkcjach oraz korzystanie z nowszych bibliotek systemów operacyjnych) muszą zostać włączone do bibliotek podstawowych funkcji, które są używane w bieżącym projekcie i przyszłych projektach.

Ukierunkowanie na wiele funkcji, które działają na tym samym typie elementu GUI w celu konsolidacji

Dużą część tego, co ma miejsce podczas zautomatyzowanego przebiegu testów, stanowi sprawdzenie elementów GUI. To sprawdzenie służy do dostarczenia informacji o elementach GUI (np. widoczna/niewidoczna, włączona/niewłączona, rozmiar i wymiary oraz dane). Dzięki tym informacjom test automatyczny może wybrać element z listy rozwijanej, wprowadzić dane do pola i odczytać wartość z pola. Istnieje kilka funkcji, które mogą działać na elementy sterujące w celu uzyskania tych informacji. Niektóre funkcje są niezwykle wyspecjalizowane, podczas gdy inne mają charakter bardziej ogólny. Może np. istnieć określona funkcja, która działa tylko na listach rozwijanych. Alternatywnie może istnieć funkcja, która działa z kilkoma funkcjami, określając funkcję jako jeden z jej parametrów. W związku z tym TAE może korzystać z kilku funkcji, które można skonsolidować do mniejszej liczby funkcji, osiągając te same wyniki i minimalizując utrzymanie.

Refaktoryzacja TAA w celu uwzględnienia zmian w SUT

W całym cyklu życia TAS konieczne będzie wprowadzanie zmian, aby uwzględniać zmiany w SUT. W miarę ewolucji i dojrzewania SUT podstawowa TAA będzie musiała również ewoluować, aby zapewniać zdolność do wspierania SUT. Należy zachować ostrożność podczas rozszerzania funkcji, aby nie były one wdrażane jako dodatkowe, ale zamiast tego były analizowane i wprowadzane poprzez zmiany w TAA. Dzięki temu, gdy nowa funkcjonalność SUT będzie wymagać dodatkowych skryptów testowych, zostaną wprowadzone kompatybilne komponenty, aby dostosować się do tych nowych testów automatycznych.

Standaryzacja i konwencje nazewnictwa

W miarę wprowadzania zmian konwencje nazewnictwa nowego kodu automatyzacji testów i bibliotek funkcji muszą być spójne z wcześniej zdefiniowanymi standardami (patrz sekcja 4.3.1).

Ocena istniejących skryptów testowych do eliminacji/rewizji SUT

Proces zmian i udoskonaleń obejmuje również ocenę istniejących skryptów testowych, ich wykorzystanie i ciągłą wartość. Jeśli na przykład niektóre testy są złożone i czasochłonne, dzielenie ich na mniejsze testy może być bardziej opłacalne i wydajne. Skierowanie testów, które są przeprowadzane rzadko lub wcale, do wyeliminowania, zmniejszy złożoność TAS i przyniesie większą jasność tego, co należy utrzymać.

8.1.4 Kandydat podsumowuje możliwości korzystania z narzędzi do automatyzacji testów

Oprócz rzeczywistego testowania automatyzacja testów może pomóc w niespecyficznych czynnościach testowych, takich jak:

Konfiguracja i kontrola środowiska

Niektóre skrypty testowe (np. tworzenie danych testowych) można wykorzystać w metodzie konfiguracji do tworzenia różnych danych testowych w nowym środowisku testowym. W sytuacji, gdy użytkownicy z wieloma profilami muszą być tworzeni na podstawie różnych danych wejściowych, zespół może użyć zautomatyzowanego skryptu testowego, aby wywołać punkt końcowy usługi internetowej, który rejestruje tych użytkowników. Skrypty testowe mogą mieć kontrolę nad konfiguracją infrastruktury testowej i można je wykorzystać do czyszczenia po zakończeniu procesu. Pomaga to zaoszczędzić czas i zapewnić, aby odpowiedni użytkownicy byli obecni w każdym nowym środowisku testowym. Na przykład różne dzienniki testów i inne testalia mogą być automatycznie usuwane ze środowiska testowego, dzięki czemu utrzymanie porządku i korzystanie ze środowiska testowego jest bardziej wydajne.

Starzenie się danych

Automatyzacja testów może być używana do manipulowania danymi testowymi w środowisku testowym. Np. w bazach danych pola daty można sprawdzać i kontrolować, aby były aktualne z perspektywy roku.

Generowanie zrzutów ekranu i filmów

Większość nowoczesnych narzędzi do automatyzacji testów interfejsu użytkownika ma wbudowaną możliwość tworzenia zrzutów ekranu lub filmów pod pewnymi warunkami i przechowywania ich. Dzięki tym narzędziom testowym zespoły mogą wspierać firmę w tworzeniu rzeczywistych zrzutów ekranu i filmów na potrzeby dokumentacji wydania oprogramowania lub do celów marketingowych.

9 Bibliografia

Normy

Normy automatyzacji testów obejmują między innymi:

The Automatic Test Markup Language (ATML) by IEEE (Institute of Electrical and Electronics Engineers), w tym: • IEEE 1671.1: Test Description • IEEE 1671.2: Instrument Description • IEEE 1671.3: UUT Description • IEEE 1671.4: Test Configuration Description • IEEE 1671.5: Test Adaptor Description • IEEE 1671.6: Test Station Description • IEEE 1641: Signal and Test Definition IEEE 1636.1: Test Results
ISO/IEC/IEEE 29119-5 (2016) Software and systems engineering – Software testing – Part 5: Keyword Driven Testing
ISO/IEC 30130:2016 (E) Software engineering — Capabilities of software testing tools The Testing and Test Control Notation (TTCN-3) opracowana przez ETSI (European Telecommunication Standards Institute) i ITU (International Telecommunication Union), w tym: • 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: Usign XML with TTCN-3 • ES 201 873-10: Documentation 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
Profil testowy UML (UTP) firmy OMG (Object Management Group) określający koncepcje specyfikacji testów w odniesieniu do: • Architektury testów • Danych testowych • Zachowania testowego • Rejestrowania testów • Zarządzania testami

Dokumenty ISTQB®

Identyfikator	Materiały referencyjne
ISTQB-AL-TTA	Certyfikowany tester ISTQB, sylabus Poziomu Zaawansowanego, Techniczny Analityk Testów, wersja 4.0, czerwiec 2021 r., dostępny w [ISTQB-Web]
ISTQB-FL	Certyfikowany tester ISTQB, sylabus Poziomu Podstawowego, wersja 4.0, kwiecień 2023 r., dostępny w [ISTQB-Web]
ISTQB-MBT	Certyfikowany tester ISTQB, sylabus Poziomu Specjalista, Testowanie oparte na modelu, wersja 1.0, październik 2015 r., dostępny w [ISTQB-Web]
ISTQB-PT	Certyfikowany tester ISTQB, sylabus Poziomu Specjalista, Tester Wydajności, grudzień 2018 r., dostępny w [ISTQB-Web]
ISTQB-SEC	Certyfikowany tester ISTQB, sylabus Poziomu Specjalista, Tester Zabezpieczeń, wersja 1.0, marzec 2016 r., dostępny w [ISTQB-Web]
ISTQB-TAS	Certyfikowany tester ISTQB, sylabus Poziomu Specjalista, Strategia Automatyzacji Testów, luty 2024 r., dostępny w [ISTQB-Web]
ISTQB – słownik	Słownik terminów testowych ISTQB, dostępny online w [ISTQB-Web]

Książki

Paul Baker, Zhen Ru Dai, Jens Grabowski i Ina Schieferdecker, „Model-Driven Testing: Using the UML Testing Profile”, Springer 2008, ISBN-10: 3540725628, ISBN-13: 978-3540725626
Efriede Dustin, Thom Garrett, Bernie Gauf, „Implementing Automated Software Testing: how to save time and lower costs while raising quality”, Addison-Wesley, 2009, ISBN 0-321-58051-6
Efriede Dustin, Jeff Rashka, John Paul, „Automated Software Testing: introduction, management, and performance”, Addison-Wesley, 1999, ISBN-10: 0201432870, ISBN-13: 9780201432879
Mark Fewster, Dorothy Graham, „Experiences of Test Automation: Case Studies of Software Test Automation”, Addison-Wesley, 2012
Mark Fewster, Dorothy Graham, „Software Test Automation: Effective use of test execution tools”, ACM Press Books, 1999, ISBN-10: 0201331403, ISBN-13: 9780201331400
Robert C Martin, „Clean Code: A Handbook of Agile Software Craftsmanship”, 2008, ISBN-10: 9780132350884
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
Daniel J. Mosley, Bruce A. Posey, „Just Enough Software Test Automation”, Prentice Hall, 2002, ISBN- 10: 0130084689, ISBN-13: 9780130084682
Manikandan Sambamurthy, „Test Automation Engineering Handbook”, styczeń 2023 r., ISBN: 9781804615492
Colin Willcock, Thomas Deiß, Stephan Tobies i Stefan Keil, „An Introduction to TTCN-3”, Wiley, wydanie drugie 2011, ISBN-10: 0470663065, ISBN-13: 978-0470663066

Artykuły

Robert V. Binder, Suzanne Miller, „Five Keys to Effective Agile Test Automation for Government Programs”, 24 sierpnia 2017 r., Software Engineering Institute, Carnegie Mellon University, https://resources.sei.cmu.edu/asset_files/Webinar/2017_018_101_503516.pdf
DoD CIO, Modern Software Practices „DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools”, wersja 2.2, maj 2023 r., https://dodcio.defense.goV/Portals/0/Documents/Library/DevSecOpsActivitesToolsGuidebookTables.pdf?ver=_Sylg1WJB9K0Jxb2XTvzDQ%3d%3d
Péter Foldházi, „Tri-Layer Testing Architecture”, https://www.pnsqc.org/docs/PROP53522057-FoldhaziDraftFinal.pdf
Thomas Pestak, William Rowell, PhD, „Automated Software Testing Practices and Pitfalls”, wrzesień 2018 r., https://www.afit.edu/stat/statcoe_files/Automated%20Software%20Testing%20Practices%20and%20Pitfalls%20Rev%201.pdf
Jim Simpson, Jim Wisnowski, „Automated Software Testing Implementation Guide”, kwiecień 2017 r., https://www.afit.edu/stat/statcoe_files/Automated%20Software%20Testing%20Implementation%20Guide.pdf
The Selenium Project, „Page object models”, https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/

10 Załącznik A – Cele nauczania i poziomy poznawcze

Poniżej przedstawiono cele nauczania obowiązujące w przypadku niniejszego sylabusa. Znajomość każdego z zagadnień poruszonych w sylabusie będzie sprawdzana na egzaminie zgodnie z przypisanym celem nauczania.

Opis celu nauczania zawiera czasownik wyrażający czynność, który odpowiada właściwemu poziomowi wiedzy wymienionemu poniżej.

Poziom 2: zrozumieć (K2)

Kandydat potrafi uzasadnić lub wyjaśnić stwierdzenia dotyczące danego zagadnienia, a także podsumować, porównać i sklasyfikować pojęcia z zakresu testowania oraz podać odpowiednie przykłady.

Czasowniki określające czynność: klasyfikować, porównywać, rozróżniać, wyjaśniać, podawać przykłady, interpretować, podsumowywać

Przykłady	Uwagi
Kandydat potrafi sklasyfikować narzędzia testowe zgodnie z ich przeznaczeniem i wspieranymi przez nie czynnościami testowymi.	
Kandydat potrafi porównać poszczególne poziomy testów.	Może służyć do wyszukiwania podobieństw, różnic lub obu.
Kandydat potrafi odróżnić testowanie od debugowania.	Szuka różnic między koncepcjami.
Kandydat potrafi rozróżnić ryzyka projektowe i produktowe.	Umożliwia oddzielne klasyfikowanie dwóch (lub więcej) pojęć.
Kandydat potrafi wyjaśnić wpływ kontekstu na proces testowy.	
Kandydat potrafi podać przykłady powodów, dla których testowanie jest konieczne.	
Kandydat wyciąga wnioski o podstawowej przyczynie defektu na podstawie danego profilu awarii.	
Kandydat potrafi podsumować czynności wykonywane w ramach procesu przeglądu produktów pracy.	

Poziom 3: zastosować (K3)

Kandydat potrafi wykonać odpowiednią procedurę, gdy zostanie mu postawione znane mu zadanie, bądź wybrać właściwą procedurę i zastosować ją w danym kontekście.

Czasowniki określające czynność: zastosować, zaimplementować, przygotować, użyć.

Przykłady	Uwagi
Kandydat potrafi stosować analizę wartości brzegowych do tworzenia przypadków testowych na podstawie danych wymagań.	Powinno odnosić się do procedury/techniki/procesu itp.

Kandydat potrafi wdrożyć metody gromadzenia metryk w celu spełnienia wymagań technicznych i zarządczych.	
Kandydat potrafi przygotować testy instalowalności aplikacji mobilnych.	
Kandydat potrafi wykorzystać śledzenie wymagań do monitorowania postępów testów pod kątem kompletności i spójności z celami testów, strategią testów i planem testów.	Może być stosowane w LO, w którym kandydat musi być w stanie zastosować technikę lub procedurę. Podobne do „stosować”.

Poziom 4: przeanalizować (K4)

Kandydat potrafi rozdzielić informacje związane z procedurą lub techniką na ich części składowe w celu lepszego zrozumienia i potrafi odróżnić fakty od wniosków. Typowym zastosowaniem jest analiza dokumentu, oprogramowania lub sytuacji projektowej i zaproponowanie odpowiednich działań w celu rozwiązania problemu lub wykonania zadania.

Czasowniki określające czynność: analizować, dekonstruować, nakreślać, ustalać priorytety, wybierać

Przykłady	**Uwagi**
Kandydat potrafi przeanalizować daną sytuację projektową, aby określić, które techniki testowania czarnoskrzynkowego lub w oparciu o doświadczenie należy zastosować, aby osiągnąć określone cele.	Może być przedmiotem egzaminu tylko w połączeniu z mierzalnym celem analizy. Powinno mieć formę „Przeanalizuj xxxx do xxxx” (lub podobną).
Kandydat potrafi nadawać priorytety przypadkom testowym w danym zestawie testowym do wykonania w oparciu o powiązane ryzyka produktowe.	
Kandydat potrafi wybrać odpowiednie poziomy i typy testów w celu weryfikacji danego zestawu wymagań.	Potrzebne, gdy wybór wymaga analizy.

Materiały referencyjne

(w zakresie poziomów poznawczych celów nauczania)

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

11 Załącznik B - Macierz powiązań między celami biznesowymi a celami nauczania

Ten rozdział zawiera powiązania między celami biznesowymi a celami nauczania dla sylabusa Inżynieria Automatyzacji Testów.

Cele biznesowe – Inżynieria Automatyzacji Testów			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12
Rozdział 1	Wprowadzenie i cele automatyzacji testów													
1.1	Opisywanie celu automatyzacji testów													
1.1.1	Kandydat wyjaśnia zalety i wady automatyzacji testów.	2	X											
1.2	Automatyzacja testów w cyklu wytwarzania oprogramowania													
1.2.1	Kandydat wyjaśnia, w jaki sposób automatyzacja testów jest stosowana w różnych modelach cyklu wytwarzania oprogramowania.	2		X										
1.2.2	Kandydat wybiera odpowiednie narzędzia do automatyzacji testów w odniesieniu do danego systemu podlegającego testowaniu.	2		X										
Rozdział 2	Przygotowanie do automatyzacji testów													
2.1	Potrzeby konfiguracyjne infrastruktury i środowisk wytwarzania													
2.1.1	Kandydat opisuje potrzeby konfiguracyjne infrastruktury umożliwiające wdrożenie automatyzacji testów.	2			X									
2.1.2	Kandydat wyjaśnia, w jaki sposób automatyzacja testów jest wykorzystywana w różnych środowiskach.	2			X									
2.2	Proces oceny w celu wyboru odpowiednich narzędzi i strategii													
2.2.1	Kandydat analizuje system podlegający testowaniu w celu określenia odpowiedniego rozwiązania dla testów automatycznych.	4				X								
2.2.2	Kandydat ilustruje wyniki techniczne oceny narzędzia.	4				X								
Rozdział 3	Architektura testów automatycznych													

Cele biznesowe – Inżynieria Automatyzacji Testów			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12
3.1	Koncepcje projektowe wykorzystywane w automatyzacji testów													
3.1.1	Kandydat wyjaśnia główne możliwości w architekturze testów automatycznych.	2					X							
3.1.2	Kandydat wyjaśnia, jak zaprojektować rozwiązanie dla testów automatycznych.	2					X							
3.1.3	Kandydat stosuje warstwowanie struktur do testów automatycznych.	3					X							
3.1.4	Kandydat stosuje różne podejścia do automatyzacji przypadków testowych.	3					X							
3.1.5	Kandydat stosuje zasady projektowania i wzorce projektowe w automatyzacji testów.	3					X							
Rozdział 4	Wdrażanie testów automatycznych													
4.1	Rozwój automatyzacji testów													
4.1.1	Kandydat stosuje wytyczne, które wspierają skuteczne działania pilotażowe i wdrożeniowe w zakresie automatyzacji testów.	3						X						
4.2	Ryzyko związane z rozwojem automatyzacji testów													
4.2.1	Kandydat analizuje ryzyka związane z wdrażaniem i planuje strategię łagodzenia skutków w zakresie automatyzacji testów.	4							X					
4.3	Utrzymywalność rozwiązania dla testów automatycznych													
4.3.1	Kandydat wyjaśnia, które czynniki wspierają i wpływają na utrzymywalność rozwiązania dla testów automatycznych.	2								X				
Rozdział 5	Wdrażanie i strategie wdrażania automatyzacji testów													
5.1	Integracja z potokami CI/CD													
5.1.1	Kandydat stosuje automatyzację testów na różnych poziomach testów w potokach.	3									X			
5.1.2	Kandydat wyjaśnia zarządzanie konfiguracją testaliów.	2									X			

Cele biznesowe – Inżynieria Automatyzacji Testów			B01	B02	B03	B04	B05	B06	B07	T08	B09	B10	B11	B12
5.1.3	Kandydat wyjaśnia zależności automatyzacji testów dla infrastruktury API.	2									X			
Rozdział 6	Raportowanie i metryki automatyzacji testów													
6.1	Gromadzenie, analiza i raportowanie danych automatyzacji testów													
6.1.1	Kandydat stosuje metody gromadzenia danych z rozwiązania dla testów automatycznych i systemu.	3										X		
6.1.2	Kandydat analizuje dane z rozwiązania dla testów automatycznych i systemu podlegającego testowaniu w celu lepszego zrozumienia wyników testów.	4										X		
6.1.3	Kandydat wyjaśnia, w jaki sposób sporządza się i publikuje raport o postępie testów.	2										X		
Rozdział 7	Weryfikacja rozwiązania dla testów automatycznych													
7.1	Weryfikacja infrastruktury testów automatycznych													
7.1.1	Kandydat planuje weryfikację środowiska automatyzacji testów, w tym konfigurację narzędzia testowego.	3											X	
7.1.2	Kandydat wyjaśnia prawidłowe zachowanie danego skryptu testowego dla testów automatycznych i/lub zestawu testów.	2											X	
7.1.3	Kandydat identyfikuje miejsca, w których testy automatyczne dają nieoczekiwane wyniki.	2											X	
7.1.4	Kandydat wyjaśnia, w jaki sposób analiza statyczna może pomóc w testowaniu jakości kodu automatyzacji.	2											X	
Rozdział 8	Ciągłe doskonalenie													
8.1	Możliwości ciągłego doskonalenia testów automatycznych													
8.1.1	Kandydat poznaje możliwości poprawy przypadków testowych poprzez gromadzenie i analizę danych.	3												X
8.1.2	Kandydat analizuje aspekty techniczne wdrożonego rozwiązania dla testów automatycznych i przedstawia zalecenia dotyczące udoskonaleń.	4												X

Cele biznesowe – Inżynieria Automatyzacji Testów			B01	B02	B03	B04	B05	B06	B07	T08	B09	B10	B11	B12
8.1.3	Kandydat przeprowadza restrukturyzację testaliów do testów automatycznych w celu ich dostosowania do aktualizacji SUT.	3												X
8.1.4	Kandydat podsumowuje możliwości korzystania z narzędzi do automatyzacji testów.	2												X

12 Załącznik C - Opis wydania

Sylabus Inżynieria Automatyzacji Testów ISTQB® 2024 to duża aktualizacja i przebudowa wersji z 2016 roku. W związku z tym nie sporządzono szczegółowego opisu wydania w podziale na rozdziały i podrozdziały, a treść została znacznie ulepszona pod kątem roli inżyniera automatyzacji testów, podczas gdy treść strategiczna została przeniesiona do nowego sylabusa ISTQB® dotyczącego strategii automatyzacji testów 2024.

13 Załącznik D – terminy specjalistyczne

Termin	Definicja
DevSecOps	Metodologia, która łączy rozwój, zabezpieczenia i operacje oraz wykorzystuje wysoki stopień automatyzacji.
wzór modelu przepływu	Ogólny obraz domeny pracy, jej komponentów i wzajemnych powiązań między nimi.
podróż użytkownika	Seria kroków, które pokazują, z perspektywy doświadczenia danej osoby, w jaki sposób użytkownik wchodzi w interakcję z systemem podlegającym testowaniu.
wzór obiektu strony	Wzór projektowy w automatyzacji testów, który ma na celu poprawę utrzymania testów i ograniczenie powielania kodu.
kontrola wersji	Proces sprawdzania i przechowywania określonych wersji kodu źródłowego.

14 Indeks

Słowa kluczowe są zdefiniowane w Słowniku terminów testowych ISTQB® (http://glossary.istqb.org/).