**Ad Usum Internum**

**Nie za krótki poradnik dla testera**

**Czyli co każdy techniczny tester powinien wiedzieć aby zabłysnąć na rozmowie kwalifikacyjnej i nie tylko!**

*“In this World there are two kinds of software testers my friends, those who collect certifications and those who think outside the box.”*

*Tuco*

Ver. 1.0

Opracowanie: Przemysław Jagodziński

Konsultacje: Dariusz Szudrzyński

**Spis Treści:**

1. Java jako idealny przykład języka obiektowego
2. REST - podstawy architektury
3. Technologie i frameworki developerskie
4. Continuous Integration - kroki i proces
5. GIT komendy i VCS
6. Teoria testowa
7. Trochę o automatyzacji i technicznych spraw ciąg dalszy
8. SQL
9. Komendy bash
10. Agile
11. Pytania otwarte
12. Saturn V - budujemy rakietę

|  |  |
| --- | --- |
| **﻿1. Java jako idealny przykład języka obiektowego** | |
| **1.1 Trzy zasady programowania obiektowego** | Hermetyzacja - Ograniczenie dostępu do wnętrza programu przez koncepcję klasy, ze względów bezpieczeństwa lub czysto praktycznych.  Dziedziczenie - Proces w którym jeden obiekt otrzymuje właściwości innego obiektu.  Polimorfizm -To cecha, dzięki której jeden interfejs może być stosowany do wykonania różnych zadań. |
| **1.2 Typy danych** | Proste/podstawowe  Całkowite  Zmiennoprzecinkowe  Znakowe  Logiczne |
| **1.3 Instrukcje sterujące** | Instrukcje wyboru  Konstrukcja switch |
| **1.4 Instrukcje iteracji** | Pętla while  Pętla do-while  Pętla for  Pętla for-each |
| **1.5 Instrukcje skoku:** | Instrukcja break  Instrukcja continue  Instrukcja return |
| **1.6 Klasa** | To konstrukcja logiczna, zbiór metod zmiennych itp. które definiują obiekt. Ważne klasa definiuje nowy typ danych. |
| **1.7 Obiekt** | To zastosowanie klasy, wystąpienie klasy w praktyce. |
| **1.8 new** | Operator, który alokuje pamięć do działania obiektu i zwraca referencję do niego. |
| **1.9 Konstruktor** | metoda o tej samej nazwie co klasa. inicjalizuje obiekt zaraz po jego utworzeniu. Czyli inicjalizuje wszystkie metody i zmienne w klasie. |
| **1.10 this** | Operator stosuje się w:  Chaining - chcemy wywołać ze zmiennej obiektowej metody, które mają typ obiektu  Parametryzacja - odróżnienie zmiennej lokalnej od parametru o tej samej nazwie |
| **1.11 Metoda przeciążona** | Metoda o tej samej nazwie co istniejąca ale o innej liczbie i typie parametrów. Może zwracać też inny typ wartości. Plus jest taki, że możemy wykorzystywać wielokrotnie tę samą nazwę metody. |
| **1.12 Metoda rekurencyjna** | Metoda wywołująca samą siebie. |
| **1.13 static** | Zmienna statyczna - to zmienna globalna współdzielona przez wszystkie obiekty klasy.  Metoda statyczna - może wywoływać jedynie inne metody statyczne, ma dostęp jedynie do danych statycznych i nie można korzystać z this i super. |
| **1.14 final** | Zmienne - uniemożliwia modyfikację zmiennej.  Metody - uniemożliwia przesłanianiu i dziedziczeniu. |
| **1.15 super (uwaga w C# nie istnieje)** | Można wywołać konstruktor klasy bazowej.  Można wywołać, uzyskać dostęp do np. metoda klasy nadrzędnej, które zostały przesłonięte przez klasę pochodną. |
| **1.16 Metoda przesłonięta** | To taka metoda, która ma taką samą nazwę co metoda klasy bazowej. Zostanie ona wykonana. A ta z klasy bazowej ukryta. |
| **1.17 Klasa abstrakcyjna** | Klasa, która stanowi w pewnym sensie jedynie strukturę dla klas podrzędnych. Dopiero klasy dziedziczące implementują np. metody. |
| **1.18 Pakiet** | Przestrzeń nazw i sterowanie dostępem. Innymi słowy w pakiecie może istnieć tylko jedna klasa o danej nazwie. |
| **1.19 Interfejs** | To coś jak klasa abstrakcyjna ale ma jedną dużą zaletę, klasa może implementować dowolną liczbę interfejsów |
| **1.20 Modyfikatory dostępu** | private - mamy dostęp jedynie w klasie, w której obiekt został zdefiniowany  brak modyfikatora - mamy dostęp w klasie, w której został obiekt zdefiniowany, w  podklasach i innych klasach ale dla danego pakietu  protected - to samo co w przypadku braku modyfikatora + podklasy z innego pakietu  public - dostępny w klasach i podklasach we wszystkich pakietach |
| **1.21 Typy wyjątków** |  |
| **1.22 Klauzula try catch** |  |
| **1.23 Garbage Collection** | Jak już wiemy np. każdemu nowemu obiektowi przydzielana jest pamięć. W javie pamięcią i jej zwalnianiem nie zajmuje się programista, ale specjalna funkcja/metoda zwana garbage collector. |
| **1.24 Kolekcje w Javie** | Lista  Zbiór  Mapa |
|  | |
| **2. REST - podstawy architektury** | |
| **2.1 Czym jest REST?** | To zestaw metod i dobrych praktyk definiujących sposób komunikacji między systemami. Może to być np. komunikacja Klient - Serwer. Komunikacja odbywa się przez protokół HTTP. |
| **2.2 Czym jest API?** | To samo co powyżej ale ogólnie. Zestaw protokołów, rutyn, narzędzi. |
| **2.3 Serwer** | Utożsamiany z backendem program, który świadczy usługi dla innych programów. Komunikuje się z bazą i klientem. |
| **2.4 Klient** | Program komputerowy występujący w roli klienta usług dostarczanych przez serwer. |
| **2.5 Po co REST API?** | Ustandaryzowanie zasad tak by każdy ze swoją technologią mógł się komunikować z "obcą" technologią potencjalnie niekompatybilną. |
| **2.6 Metody HTTP** | GET - request danych ze źródła  POST - update danych w źródle  PUT - zapisanie danych w źródle  DELETE - usunięcie danych  OPTIONS  HEAD |
| **2.7 Statusy** | 200 - OK  201 - Created  500 - Internal Server Error  400 - Bad request  401 - Unauthorized  404 - Not found |
| **2.8 Co testować?** | Backend validation - walidacja pól  Uwierzytelnianie - czy endpointy są zabezpieczone przed nieporządanym dostępem |
| **2.9 CORS requests** | Mechanizm umożliwiający współdzielenie zasobów pomiędzy serwerem i klientem znajdującymi się w różnych domenach, czyli wykonywania żądań AJAXowych przy zachowaniu pewnych ograniczeń co do dopuszczalnego źródła danych. |
| **2.10 Headers** | Nagłówki to pewnego rodzaju polecenia służące do komunikacji między serwerem a klientem np.  Access-Control-Allow-Headers  Access-Control-Allow-Origin  Access-Control-Allow-Methods |
|  | |
| **3. Technologie i frameworki developerskie** | |
| **3.1 Maven** | Build Manager (java) |
| **3.2 Gradle** | Build Manager (java) |
| **3.3 npm** | Package Manager (js) |
| **3.4 gulp** | Build Manager (js) |
| **3.5 NuGet** | Package Manager (C#) |
| **3.6 Proxy** | Serwer pośredniczący w przekazywaniu danych w sieci komputerowej z wykorzystaniem jakiegoś protokołu np. HTTP. |
| **3.7 curl** | Programik do transferowania danych różnymi protokołami np. HTTP. |
| **3.8 SSIS** | Paczka ETL-owa w SQL Server pozwalająca na wyciąganie danych, konwertowanie, procesowanie i załadowanie do bazy. |
| **3.9 Tomcat** | Serwer aplikacyjny. |
| **3.10 Hibernate** | Framework pod Javę do mapowania obiektów relacyjnej bazy danych. |
| **3.11 Spring** | Framework javowy. |
| **3.12 NoSQL** | Nierelacyjna baza danych. Np. MongoDB. |
| **3.13 Ajax** | Technologia wspierająca dynamiczne zmiany elementów na stronie bez konieczności przeładowywania strony. Na technikę składa się xml, javascript, XMLHttpRequest. |
| **3.14 Flask** | Framework pythonowy. |
| **3.15 Angularjs** | Js framework. Najważniejszymi składowymi aplikacji angularowej są:  Routing - zarządza przekierowaniami  Controller - pośrednik między kodem HTML a serwisem  Directive - zestaw funkcji do wykonania. ng-repeat, ng-show lub customowe  Service - serwis służy do komunikacji z backendem  Scope - odwzorowanie HTML'a na js? |
| **3.16 JDBC** | API pod Javę określające sposób komunikacji Backend-DB. Inne np. ODBC dla C#. |
| **3.17 XMLHttpRequest** | Asynchroniczne przesyłanie danych dzięki czemu w trakcie pobierania danych użytkownik może wykonywać inne czynności. |
| **3.18 node.js** | Środowisko uruchomieniowe dla JS. |
| **3.19 JRE** | Środowisko uruchomieniowe dla Javy |
| **3.20 JDK** | Środowisko do programowania w Javie. Zawiera kompilatory Javy, wirtualne maszyny itd. |
| **3.21 JQuery** | Biblioteka JS-owa. |
| **3.22 TypeScript** | Microsoftowy język obiektowy. |
| **3.23 Ant** | Build Manager (java) |
|  | |
| **4. Continuous Integration - kroki i proces** | |
| **4.1 Po co?** | Szybka możliwość weryfikacja i przetestowanie zmian wrzucanych na środowisko połączona z integracją z innymi systemami oraz możliwość zaprezentowania na demo. |
| **4.2 Konfiguracja** | Określenie nazwy builda i build countera  Zintegrowanie joba ze zdalnym repozytorium    Określenie build stepów i gdzie powinny się odpalać    Określenie triggerów  Określenie w jakich warunkach job powinien failować i jeszcze dodatkowe parametry jak odpalanie dodatkowych skryptów po buildzie. |
| **4.3 Night build** | Zazwyczaj pełna lista testów jest odpalana. Job zaskedżulowany. |
| **4.4 Daily build** | Przetestowanie korowych funkcjonalności po każdym komicie. |
| **4.5 Narzędzia CI** | Jenkins, TeamCity, CruiseControl, Bamboo |
|  | |
| **5. GIT komendy i VCS** | |
| **5.1 Narzędzia VCS** | GIT, Perforce, SVN |
| **5.2 Klienci VCS i wtyczki** | Stash, Bitbucket, GitLab, Tortoise |
| **5.3 git config** | Konfiguracja np. Credentiali, wielkości danych puszowanych. |
| **5.4 git init** | Inicjalizuje lokalne repozytorium. |
| **5.5 git clone** | Klonuje zawartość zdalnego repozytorium do lokalnego. |
| **5.6 git add** | Umożliwia dodanie plików do zakomitowania. |
| **5.7 git commit** | Komituje pliki do spuszowania. |
| **5.8 git push origin master** | Wysyła pliki na zdalne repo z master brancha lokalnego na zdalnego. |
| **5.9 git status** | Pokazuje, na którym branchu jesteśmy. |
| **5.10 git remote add origin** | Powiązanie ze zdalnym repozytorium. |
| **5.11 git checkout <branchname>** | Tworzy nowego brancha lub przeskakuje na istniejącego. |
| **5.12 git branch** | Tworzy nowego brancha. |
| **5.13 git branch -d <branchname>** | Usuwa istniejącego brancha. |
| **5.14 git pull** | Zaciąga dane ze zdalnego repozytorium. |
| **5.15 git merge <branchname>** | Łączy dane ze zdalnego brancha z danymi na lokalnym branchu. |
| **5.16 git diff** | Pokazuje różnice między commitami. |
| **5.17 git tag 1.0.0 <commitID>** | Oznaczenie komita. |
| **5.18 git log** | Pokazuje historię commitów. |
| **5.19 git fetch** | git fetch = git pull + git merge |
| **5.20 git reset --hard <branchname>** | Przeskakuje do ostatniego zakomitowanego brancha. |
| **5.21 git grep "string"** | Szukamy określonego wzorca w lokalnym repo. |
| **5.22 git stash** | Wyrzuca dane do schowka. |
| **5.23 git stash pop** | Przywraca dane ze schowka. |
| **5.24 git pull --rebase** | Jak nie chcemy merdżować kodu? |
| **5.25 git fork** | Tworzy kopię istniejącego projektu. |
|  | |
| **6. Teoria testowa** | |
| **6.1 Test Strategy** | Typowy dokument powinien zawierać: Scope and Objectives, Roles and Responsibilities, Test deliverables, Testing metrics - stworzony przez Project Managera. High Level document. |
| **6.2 Test Plan** | Typowy dokument powinien zawierać: Test Scope, Type of tests, Entry Criteria, Exit Criteria, Bugs definitions, Environments, Testing approach (manual, automated) - Lower level document. |
| **6.3 Traceability Matrix** | Pokrycie wymagań testami. |
| **6.4 Test Scenario** | Jest pojęciem szerszym niż przypadek testowy. Może zawierać wiele przypadków testowych. To po prostu procedura testowa na jakąś funkcjonalność. |
| **6.5 Test Case** | Zbiór kroków do odtworzenia, oczekiwanych rezultatów, a także krótkiego opisu testu. |
| **6.6 Podejście Testowe** | CRUD |
| **6.7 Wyrocznia Testowa** | Źródło dostarczające oczekiwanych rezultatów umożliwiające porównanie ich z otrzymanymi rezultatami. Wyrocznią może być istniejący system podręcznik użytkownika, wiedza testera, ale nie może być nią kod. |
| **6.8 Testing Metrics** | Pokrycie wymagań, liczba defektów znalezionych (critical, blockers, major) i naprawionych, czas potrzebny na naprawę defektu. |
| **6.9 SDLC** | Analiza, planowanie, Zbieranie wymagań, Development i Testing, Utrzymanie, Czynności zamykające projekt. |
| **6.10 Waterfall** | Każdy krok rozwoju aplikacji następuje po sobie. Wady dłużej trwa projekt, ciężko o wprowadzanie zmian. Zalety: bardziej uporządkowany, istnieje dokumentacja. |
| **6.11 Cykl życia defektu** | Report, assign, fix, resolve, test, approve or reject and assign, close. |
| **6.12 Raportowanie defektu** | Title, environment, sometimes sprint number, steps to reproduce, expected result ,actual result, screenshot. |
| **6.13 Techniki testowe** | Klasy równoważności - zbiór danych, których wykorzystanie powoduje takie samo działanie systemu  Wartości graniczne - specjalny przypadek klas równoważności, badamy wartości na granicy danych zbiorów i tuż poza  Przejścia pomiędzy stanami - przypadki testowe projektowane tak by sprawdzały dozwolone i niedozwolone przejścia między stanami  Tablica decyzyjna |
| **6.14 Smoke Testy** | Testy korowych funkcjonalności. |
| **6.15 Testy Regresyjne** | Przeprowadzane po smoke testach, weryfikują czy system się nie popsuł po dokonaniu zmian, wprowadzeniu nowych funkcjonalności, poprawieniu błędów. |
| **6.16 Co automatyzować** | Korowe funkcjonalności (wykorzystywane w wielu miejscach)  Testy odpalane wielokrotnie (regresyjne, smoke testy)  Testy z dużą ilością danych testowych  Testy odpalane na wielu przeglądarkach  Funkcjonalności, które mają tendencję do psucia się |
| **6.17 White box testing** | Unit testing, zaglądamy co tam w bebechach piszczy. Weryfikujemy działanie metod, klas, procedur, tabel itd. |
| **6.18 Black box testing** | Bez zaglądania w kod. Testujemy w oparciu o techniki testowe. |
| **6.19 Ad Hoc testing** | Bez przygotowania, bazujemy jedynie na własnym doświadczeniu. |
| **6.20 Exploratory testing** | Bazujemy na własnym doświadczeniu. Przygotowujemy się do tego. Może być odrobina dokumentacji. |
| **6.21 Sesja eksploracyjna** | Session based testing (2-3 sessions) each session 30-45 minutes.  Session metrics  The session metrics are the primary means to express the status of the exploratory test process. They contain the following elements:  Number of sessions completed  Number of problems found  Function areas covered  Percentage of session time spent setting up for testing  Percentage of session time spent testing  Percentage of session time spent investigating problems |
| **6.22 TDD** | Koncepcja developmentu, w którym wpierw piszemy test pod funkcjonalność, piszemy funkcjonalność, jeśli failuje to poprawiamy funkcjonalność i raz jeszcze odpalamy test. |
| **6.23 BDD** | Rozwinięcie koncepcji TDD o dodatkową warstwę przedstawiającą testy w języku naturalnym, zrozumiałym dla biznesu. |
| **6.24 Testing** | Sprawdzenie czy wymagania zostały zaimplementowane poprawnie. |
| **6.25 QA** | Szersze pojęcie niż testing. Testing jest składową QA, ale QA dba również o odczucia klienta np. estetyka, innowacyjność a także czy jakieś standardy zostały zachowane. |
| **6.26 Waga** | Jak ważny bug jest dla biznesu. |
| **6.27 Priorytet** | Jak szybko buga rozwiązać. |
| **6.28 Poziomy testowania** | Modułowe(jednostkowe)  Integracyjne  Systemowe  Akceptacyjne |
| **6.29 Analiza Statyczna** | Testowanie modułu lub systemu na etapie pisania specyfikacji i projektowania bez uruchamiania kodu. |
| **6.30 Analiza dynamiczna** | Tu uruchamiamy kod i przeglądamy ręcznie kod. |
| **6.31 Model Wodospadowy** | W tym modelu każda faza rozwoju oprogramowania następuje po sobie: Zbieranie Wymagań -> Projektowanie -> Development -> Testing -> Utrzymanie |
| **6.32 Model V** | Rozwinięcie modelu wodospadowego, gdzie każdy etap projektowania ma swój odpowiednik po stronie weryfikacji. I tak wymagania są sprawdzane przez testy akceptacyjne architektura komponentów sprawdzana przez testy komponentów, architektura systemu sprawdzana przez testy systemowe itd. Na dole jest implementacja. |
| **6.33 Testy niefunkcjonalne** | performance, security, usability, portability, accessibility |
| **6.34 Testy obciążeniowe** | Ile requestów jest w stanie obsłużyć system w określonym czasie. |
| **6.35 Testy przeciążeniowe** | Sprawdzamy kiedy system padnie i czy padnie w oczekiwany sposób. |
| **6.36 Testy wydajnościowe** | Ile czasu trwa wykonanie jakiejś czynności, jakiegoś requesta. |
| **6.37 Sesja usability** | Testy z użytkownikami. Badamy jak długo czasu spędzał na stronie, jakie są odczucia odnośnie aplikacji, czas potrzebny na wykonanie jakiegoś zadania. |
| **6.38 Testy bezpieczeństwa** | Np. sql injection, cross site scripting. |
| **6.39 Piramida testów** | W pewnym sensie usystematyzowanie ilościowe testów. Piramida zakłada, że najwięcej odpalamy unit testów, dalej testów funkcjonalnych i akceptacyjnych. Najmniej E2E. |
| **6.40 UML** | Modelowanie obiektowe. Przykładowe diagramy:  Activity  Use cases  Classes |
| **6.41 WSDL** | (Web Services Description Language) - informacje odnośnie web-serwisu- akceptowane typy danych, port działania, operacje, informacje o end-pointach. |
| **6.42 PairWise Testing** | Czarnoskrzynkowa technika projektowania przypadków testowych, której celem jest redukcja ilości przypadków testowych w celu szybszego ukończenia testowania. Dobre zastosowanie tej techniki nie powinno obniżać pokrycia. |
| **6.43 Jarzmo testowe** | Środowisko testowe, składające się z zaślepek i sterowników potrzebnych do wykonania testu. |
| **6.44 Hot-Fix** | Poprawka wprowadzona do aplikacji będącej już na produkcji bez nowego releasu całej aplikacji. Poprawka jest wpierw testowana na środowisku akceptacyjnym będącym kopią produkcji. |
| **6.45 Change Request** | Prośba ze strony biznesu o wprowadzenie zmian do funkcjonalności aplikacji będącej na produkcji. |
| **6.46 Środowiska** | DEV - zazwyczaj lokalnie u developera ale też często współdzielone przez innych developerów.  TEST - dedykowane testerom do testów automatycznych i manualnych  UAT - środowisko przeznaczone dla klienta do testów akceptacyjnych  PROD - nie muszę chyba pisać co to oznacza :) |
|  | |
| **7. Trochę o automatyzacji i technicznych spraw ciąg dalszy** | |
| **7.1 ssh** | Protokół komunikacyjny, który umożliwia uwierzytelnianie użytkownika w trakcie komunikacji. |
| **7.2 json array** | array of objects - [{},{}] |
| **7.3 json object** | Np. { name: "name", lastName: "lastName"} |
| **7.4 Assert** | Kiedy step się wywali egzekucja testu się zatrzymuje. |
| **7.5 Verify** | Kiedy step się wywali egzekucja testu jest kontynuowana. |
| **7.6 Implicit Wait** | Taki globalny wait dla wszystkich elementów. Czekaj aż wyskoczy wyjątek, że nie można znaleźć elementu. |
| **7.7 Explicit Wait** | Taki lokalny dla konkretnego elementu wait. |
| **7.7 Fluent Wait** | Przeznaczony dla elementów, których czas pojawienia się jest bardzo zmienny. |
| **7.8 Page Object Pattern** | Odwzorowanie np. strony webowej pod postacią klasy z metodami odpowiadającymi danej funkcjonalności. Z Takiego obiektu, czyli wystąpienia klasy wywołujemy metody. |
| **7.9 Page Object Model** | Rozwinięcie albo dodatek do Page object pattern. Stworzona po to by nie powielać kodu i składować i inicjalizować web elementy w jednym miejscu.  Najprościej jest utworzyć klasę z web elementami i zainicjalizować je (wyszukać). W ten sposób nie musimy w przypadku zmiany lokatora w wielu miejscach go zmieniać.  Oszczędzamy czas i miejsce. |
| **7.10 Page Factory** | Możemy sobie sami stworzyć taki model, ale po co jak Selenium WebDriver ma to już stworzone i zoptymalizowane. Nazywa się to Page Factory. |
| **7.11 BDD w praktyce** | BDD binding in C# and Java - [Binding] and @Binding.  Przykłady frameworków:  Specflow - BDD dla C# używa atrybutów []  JBhave - BDD dla Javy używa annotacji  Cucumber - BDD dla Javy używa annotacji i co ważne sklejanie stepów z featurami używacie poprzez słówko kluczowe “glue” w klasie z runner’em testów.  Jasmine - BDD dla JS |
| **7.12 TestNG** | Framework javowy do unit testów i funkcjonalnych. Najczęściej stosowany w połączeniu z selenium.  Annotacje TestNG: @Test, @BeforeClass, @AfterClass, @BeforeSuite, @AfterSuite, @BeforeMethod, @AfterMethod |
| **7.13 Locators** | Można generalnie podzielić na dwie grupy: związane z DOM (xpath, id, name) i CSS |
| **7.14 Aplikacje Mobilne** | Native - instalowane na mobilkach z takich źródeł jak GooglePlay. Są dedykowane konkretnym urządzeniom. Mogą używać funkcjonalności telefonu np. Kamery  Hybrid - Są dostępne np. z GooglePlay ale odpalane przeglądarkowo  Web - apki odpalane przez przeglądarkę, zazwyczaj napisane w HTML5 |
| **7.15 Testowanie mobilek** | Narzędzia do testowania automatycznego urządzeń mobilnych: appium, robotium, espresso. W przypadku webowych można używać emulatorów selenium webdriver na aplikacje mobilne. |
| **7.16 Event Listenery** | Zestaw metod, które umożliwiają logowanie akcji ale jedynie tych związanych z webdriverem. |
| **7.17 Regexp** | Ciąg znaków określający szukany wzorzec. Np. .\* szukamy plików dowolnego formatu. |
| **7.18 Programowanie niskopoziomowe** | Programowanie niskiego poziomu to język zrozumiały dla peceta [język maszynowy (kod binarny)] np. assambler. |
| **7.19 Programowanie wysokopoziomowe** | Programowanie wysokiego poziomu to programowanie zrozumiałe dla ludzi np. java. |
| **7.20 Cache** | Pamięć podręczna. Jest wykorzystywana do szybszego dostępu do danych.  Np. Pamięć podręczna przeglądarki powoduje, że elementy już kiedyś załadowane ponownie nie muszą być ładowane po prostu są pobierane z pamięci podręcznej. |
| **7.21 Protractor** | Framework jsowy dedykowany dla angularJS, służący do testów funkcjonalnych. Działa na selenium webdriverze. |
| **7.22 Środowisko uruchomieniowe** | W językach wysokopoziomowych potrzebny jest specjalny interpreter, który przetłumaczy kod na postać binarną zrozumiałą dla maszyny. Przykładami są node.js i jre. |
| **7.23 Package Manager** | System do automatycznej instalacji, konfiguracji, aktualizacji i usuwania pakietów oprogramowania. |
| **7.24 Geckodriver** | Driver dla firefoxa w selenium 3.0 + |
|  | |
| **8. SQL** | |
| **8.1 Union** | Łączenie tabel o tej samej liczbie kolumn i typie. Usuwa duplikaty. |
| **8.2 Union ALL** | Łączenie tabel o tej samej liczbie kolumn i typie. Nie usuwa duplikatów. |
| **8.3 Joins** | left join - pokazuje część wspólną dwóch tabel oraz pozostałe rekordy w "lewej" tabeli.  right join - pokazuje część wspólną dwóch tabel oraz pozostałe rekordy w "prawej" tabeli.  inner join - pokazuje część wspólną tabel.  outer join - pokazuje część wspólną i nie wspólną tabel.  cross join - łączy każdy element z tabeli 1 z każdym elementem z tabeli 2. |
| **8.4 Agregacje w SQL** | AVG, SUM, MAX, MIN |
| **8.5 EXCEPT** | (SQL Server) odejmuje dwie tabele o takiej samej ilości kolumn i tych samych typach danych. |
| **8.6 MINUS** | (Oracle) odejmuje dwie tabele o takiej samej ilości kolumn i tych samych typach danych. |
| **8.7 Count** | Count(\*) - zlicza ilość rekordów po wszystkich kolumnach  Count(1) - zlicza ilość rekordów po pierwszej kolumnie |
| **8.8 Hurtownia danych** | Baza skonstruowana z innych baz relacyjnych celem szybszego dostępu do danych.  Podstawowymi elementami są: Staging area, Atomisation, Dimensions i Tabele faktów. |
| **8.9 Dynamiczny SQL** | Czyli jak przechowywać zapytania w zmiennej, odpalać je z dowolnego miejsca kodu i przechowywać zwróconą wartość w innej zmiennej. Ma słabą wydajność.  Oracle - Execute Immediate  SQL Server - exec sp\_executesql |
| **8.10 SQL hints** | W SQL Server możemy dopomóc przy egzekucji zapytania pewnymi hintami celem przyspieszenia działania zapytania.  join hints - merge, loop, hash  table hints - NO LOCK |
| **8.11 Pivot** | Obracanie tabeli. |
| **8.12 Tabela CTE** | Common table expression - służąca chociażby do insertowania w jednym zapytaniu dużej ilości rekordów. |
| **8.13 Klucze** | Primary key  Foreign key |
| **8.14 Constraints** | Określają ograniczenia kolumn: not null, unique, primary key, foreign key, index. |
| **8.15 Grupowanie** | Grupowanie należy wykonywać po niezagregowanych kolumnach. |
| **8.16 Transakcje** | Insert, update, delete. |
| **8.17 Indeksy** | Klastrone  Nieklastrowe |
| **8.18 Konwertowanie danych** | Convert - więcej możemy zrobić z datą niż w przypadku cast  Cast - jest standardem ANSI |
| **8.19 Trimming** | ltrim - wycina spację z lewej strony  rtrim - wycina spację z prawej strony |
| **8.20 Truncate** | Chyba nie zwraca uwagi na klucze. Usuwa jak leci. |
| **8.21 Delete** | Tutaj niestety errory wyskoczą, jak są indeksy i klucze pozakładane. |
|  | |
| **9. Komendy bash** | |
| **9.1 cat [filename]** | Pokazuje zawartość pliku. |
| **9.2 cd /directorypath** | Przechodzimy pomiędzy katalogami. |
| **9.3 chmod [options] mode filename** | Zmienia uprawnienia do pliku. |
| **9.4 chown [options] filename** | Zmienia właściciela pliku. |
| **9.5 clear** | Czyści konsolę. |
| **9.6 cp [options] source destination** | Kopiuje plik do katalogu. |
| **9.7 date [options]** | Pokazuje lub ustawia datę systemową. |
| **9.8 df [options]** | Pokazuje jak zajęty jest dysk twardy. |
| **9.9 du [options]** | Pokazuje ile ważą pliki. |
| **9.10 file [options] filename** | Określa jaki typ danych jest wewnątrz pliku. |
| **9.11 find [pathname] [expression]** | Znajdź pliki dla danego wzorca. |
| **9.12 grep [options] pattern [filesname]** | Pokaż pliki dla danego wzorca. |
| **9.13 kill [options] pid** | Zatrzymuje proces. kill -9 bezwarunkowo. |
| **9.14 less [options] [filename]** | Pokazuje zawartość zawartość pliku strona po stronie. |
| **9.15 ln [options] source [destination]** | Miękkie dowiązanie. |
| **9.16 ls [options]** | Wylistuj zawartość katalogu. |
| **9.17 man [command]** | Pomoc. |
| **9.18 mkdir [options] directory** | Stwórz nowy katalog. |
| **9.19 mv [options] source destination** | Zmień nazwę pliku lub wytnij i wklej go do innej lokalizacji. |
| **9.20 passwd [name [password]]** | Zmiana hasła. |
| **9.21 ps [options]** | Pokazuje działające procesy. |
| **9.22 pwd** | Ścieżka do katalogu, z którego wykonano komendę. |
| **9.23 rm [options] directory** | Usuwa pliki lub katalogi. |
| **9.24 rmdir [options] directory** | Usuwa katalogi. |
| **9.25 ssh [options] user@machine** | Zdalne łączenie się z inną linuxową maszyną. |
| **9.26 su [options] [user [arguments]]** | Przechodzenie pomiędzy użytkownikami. |
| **9.27 tail [options] [filename]** | Pokazuje ostatnich 10 linijek pliku. |
| **9.28 tar [options] filename** | Pakowanie i rozpakowanie pliku. |
| **9.29 touch filename** | Tworzy pusty plik. |
| **9.30 who [options]** | Pokazuje kto jest zalogowany. |
|  | |
| **10. Agile** | |
| **10.1 Zalety agile** | Łatwiejsza reakcja na zmiany, szybciej trwa projekt. |
| **10.2 Wady agile** | Większy chaos, dużo bugów, zero dokumentacji. |
| **10.3 Kanban** |  |
| **10.4 Ceremonie Scrumowe** | Planning  Daily Stand-up  Review i Retrospective  Backlog grooming  Scrum of Scrums  Poker planning |
| **10.5 Produkty Scrumowe** | Product backlog - lista high-levelowych wymagań.  Sprint backlog - lista tasków brana na sprint.  Burndown-chart - pokazuje progres pracy. Zależność czasu wyestymowanych tasków od czasu w sprincie.  User Story - historyjka brana na sprinta. Zazwyczaj funkcjonalność. |
| **10.6 Zespół w Scrumie** | Developer (Dev Team) - wiadomo co robi  Product Owner - zarządza backlogiem, reprezentuje interesy klienta  Scrum Master - zarządza procesem i ceremoniami scrumowymi |
| **10.7 SAFe** | Scaled Agile Framework- Podejście agilowe dla dużych projektów w którym istnieje wiele zespołów scrumowych współpracujących ze sobą. |
|  | |
| **11. Pytania otwarte** | |
| **11.1** | Developerzy nie wyrobią się z taskiem w sprincie. Co robię i jak komunikuję to klientowi? |
| **11.2** | Project Manager chce by testować funkcjonalność na produkcji. Jak reaguję na taką prośbę? |
| **11.3** | Jak motywuję podwładnych do wykonywania zadań? |
| **11.4** | Jak weryfikuję progres pracy podwładnych? |
| **11.5** | Jak zarządzam rozproszonym zespołem? |
| **11.6** | Co robię w momencie kiedy źle oszacowałem ilość pracy potrzebnej na zakończenie zadań w sprincie dla zespołu testerów. |
| **11.7** | Czy byłem kiedyś pochwalony i/lub zganiony za swoją pracę? |
| **11.8** | Jak organizuję sobie pracę w projekcie, który dopiero startuje? |

|  |  |  |
| --- | --- | --- |
| **12. Saturn V - budujemy rakietę** | | |
| Zabawmy się w budowanie rakiety Saturn V z wykorzystaniem poznanych technologii. Przetestujmy ją również w fazie budowy i w trakcie lotu.  Rys historyczny - Saturn V jest jednym z największych technologicznie dokonań człowieka w historii. Rakieta jednokrotnego użytku wysoka na 110 metrów, o średnicy 10 metrów i będąca w stanie wynieść 118 ton ładunku na niską orbitę okołoziemską. Pierwszy lot odbył się 21 grudnia 1968 roku w ramach misji Apollo 8. | | |
| **Zacznijmy od podsumowania poznanych technologii** | | |
|
| **Biznes Analiza i zarządzanie projektem** | UML, JIRA, SDLC, Waterfall, Agile, Trello, Confluence, VSTS | |
| **Baza Danych/Hurtownia Danych** | Relacyjne (SQL) | Oracle (SQL, PL/SQL), SQL Server (TSQL), MySQL, PostgreSQL |
| Nierelacyjne (NOSQL) | MongoDB, Application Insights (AIQL) |
| **Backend** | Języki | Java, C#, Python, PHP |
| Frameworki | Spring, Hibernate, .NET, Flask |
| **Frontend** | Języki | JavaScript (client), TypeScript(client), HTML (web), CSS (web) |
| Frameworki | AngularJS (client), Angular (client), React.js (client), Backbone.js (client), Bootstrap (web) |
| **API** | PODBC, JDBC, SOAP, REST, ODBC | |
| **Continuous Integration** | TeamCity, Jenkins, Bamboo, Cruise Control | |
| **Systemy Kontroli Wersji (VCS)** | GIT, Perforce, SVN | |
| **Klienci VCS** | Stash, Bitbucket, GitLab, Tortoise | |
| **Build Manager** | Maven, Gradle, Gulp, Ant | |
| **Package Manager** | Npm, NuGet | |
| **Środowisko Uruchomieniowe** | JRE, node.js | |
| **Serwer Aplikacyjny/Webowy** | Tomcat, JBoss, IIS | |
| **Frameworki testowe** | TestNG, Nunit, Specflow, Jbehave, Cucumber, Junit, RestAssured, Selenium, Appium, Robotium, Espresso, Jasmine, Protractor | |
| **Typy testów** | Jednostkowe, Jednostkowe-Integracyjne, Integracyjne, Systemowe, Akceptacyjne, Regresyjne, Smoke Testy, Testy Manualne, Testy Automatyczne, Funkcjonalne, Niefunkcjonalne, E2E | |
| **Narzędzia do testów i zarządzania testami** | SoapUI, HP QC/ALM, Testlink, VSTS, Jmeter, HP QTP/UFT | |
|  | | |
|
| **Rakieta Saturn V w klockach** | | |
|  | | |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Klocek 1**  Główny silnik  Napędza rakietę  Posiada elementy stabilizujące lot  Posiada osłonę termiczną  Posiada osłonę antykolizyjną | **Klocek 2**  Drugi silnik  Steruje rakietą na niskiej orbicie  Posiada osłonę termiczną  Posiada osłonę antykolizyjną | **Klocek 3**  Moduł centralny  Stąd astronauci zarządzają rakietą przez komunikację z centralnym komputerem  Zawiera moduł mieszkalny i kokpit  Stąd astronauci sterują rakietą  Na końcu jest przytwierdzona antena komunikacyjna | **Klocek 4**  Centralny Komputer  Steruje przepływem paliwa w silnikach  Steruje elementami podtrzymującymi życie w module centralnym  Zbiera informacje ze wszystkich czujników  Umożliwia komunikację z centrum kontroli lotów poprzez antenę komunikacyjną | **Klocek 5**  Kapsuła ratunkowa (backup system)  Gromadzi dane z centralnego komputera i w razie awarii lub katastrofy można określić co spowodowało ów failure  Umożliwia bezpieczny powrót astronautów na Ziemię w przypadku katastrofy | **Klocek 6**  Rampa Startowa  Służy do poskładania rakiety do kupy  Umożliwia start rakiety  Jest wielokrotnego użytku  Można wykonać próbny/testowy start | | | |
|  | | |
| **Korzystając z przedstawionych w tabelce technologii i członów rakiety odpowiedz na następujące pytania. Podaj przykładowe kombinacje technologii.** | | |
| 0. Podpisałeś kontrakt z NASA, zaplanuj swoją pracę, jakich narzędzi użyjesz do zaprojektowania rakiety?  UML - do zaprojektowania architektury i całego systemu, włącznie z przepływem danych  JIRA - do zarządzania projektem | | |
| 1. Nasza rakieta potrzebuje silnika, który poniesie astronautów do gwiazd. Wybierz klocki i technologie do zbudowania silnika.   Klocki: 1, 2  Technologie: Angularjs, HTML, CSS, npm, gulp | | |
| 2. Przetestuj teraz zbudowany silnik. Zaproponuj typ testów i technologie.    Testy: Jednostkowe, Funkcjonalne  Technologie: TestNG, Selenium | | |
| 3. Nasza rakieta potrzebuje czarnej skrzynki, w której składowane byłyby wszystkie informacje z komputerów pokładowych i czujników. Wybierz klocki i technologie do zbudowania silnika.  Klocki: 5  Technologie: SQL Server | | |
| 4. Przetestuj teraz czarną skrzynkę. Zaproponuj typ testów i technologie.  Testy: Jednostkowe  Technologie: TSQL | | |
| 5. Nasza rakieta potrzebuje modułu, w którym astronauci mogliby przetrwać podróż. Wybierz klocki i technologie do zbudowania takiego modułu.  Klocki: 3  Technologie: Angularjs, HTML, CSS, npm, gulp | | |
| 6. Przetestuj teraz moduł dla astronautów. Zaproponuj typ testów i technologie.  Testy: Jednostkowe, Funkcjonalne  Technologie: TestNG, Selenium | | |
| 7. Bez centralnego komputera żadna podróż się nie powiedzie. Wybierz klocki i technologie do zbudowania centralnego komputera.  Klocki: 4  Technologie: SOAP, C#, .NET, ODBC, NuGet | | |
| 8. Przetestuj go teraz. Zaproponuj typ testów i technologie.  Testy: Jednostkowe, Funkcjonalne, Systemowe, Integracyjne  Technologie: Nunit | | |
| 9. Nasza rakieta potrzebuje nowoczesnego designu. Wybierz klocki i technologie do poprawy wyglądu rakiety.  Klocki: 1, 2, 3  Technologie: bootstrap, npm, gulp | | |
| 10. Przetestuj zbudowaną rakietę jako całość. Zaproponuj typy testów i technologie.  Testy: Systemowe, Integracyjne, E2E, Testy Funkcjonalne i Niefunkcjonalne  Technologie: TestNG, Selenium | | |
| 11. Jesteś w przededniu startu rakiety. Rakieta potrzebuje miejsca, z którego mogłaby wystartować. Wybierz klocki i technologie do miejsca startowego.  Klocki: 6  Technologie: Jenkins, Tomcat | | |
| 12. Do startu zostały dni. Musimy być pewni, że NASA będzie zadowolona i rakieta nie eksploduje podczas startu. Zaproponuj typy testów i technologie.  Testy: Akceptacyjne, Testy Funkcjonalne i Niefunkcjonalne  Technologie: Testowanie Manualne | | |
| 13. Rakieta wystartowała z powodzeniem. Niestety odłamki starej radzieckiej stacji MIR spowodowały niewielkie uszkodzenie poszycia. Zaproponuj testy jakie powinno się wykonać po naprawieniu defektów.  Testy: Regresyjne | | |
| 14. Czujniki pokładowe wykryły wyciek powietrza z modułu mieszkalnego. Zaproponuj testy jakie powinno się wykonać po naprawieniu defektów.  Testy: Regresyjne | | |
| 15. Centrum Canaveral na Florydzie chce przesłać na rakietę nową wersję oprogramowania dla jednego z modułów odpowiedzialnych za działanie silników sterujących. Zaproponuj testy jakie powinno się wykonać po wdrożeniu nowej wersji oprogramowania.  Testy: Regresyjne | | |