Patronat – INTIVE

Etap 1 QA

Cele:

- Zapoznanie się z teorią testowania
- Sprawdzenie wiedzy niezbędnej do pracy w projekcie IT

1. Czym się różnią testy funkcjonalne od niefunkcjonalnych?

Testów funkcjonalnych użyjemy jeśli zależy nam na przetestowaniu jakiejś funkcji wykonywanej przez oprogramowanie. Natomiast jeśli będziemy chcieli np. przetestować niezawodność, użyteczność systemu (czyli atrybuty jakościowe) zastosujemy wówczas testowanie niefunkcjonalne.

Testowanie funkcjonalne oparte jest na analizie specyfikacji funkcjonalnej modułu lub systemu, specyfikacji wymagań, przypadkach użycia.

Natomiast **testowanie niefunkcjonalne** to testowanie atrybutów modułu lub systemu, które nie odnoszą się do jego funkcjonalności np. niezawodności, efektywności, pielęgnowalności i przenaszalności. Podczas testów niefunkcjonalnych testowane są **cechy jakościowe** części lub całości systemu. Brane są pod uwagę aspekty oprogramowania, które nie są związane z określoną funkcją lub działaniem użytkownika, takie jak skalowalność, bezpieczeństwo czy czas odpowiedzi aplikacji np. ile osób może się zalogować w jednym czasie? Testy niefunkcjonalne są wykonywane na wszystkich poziomach testów.
Testowanie funkcjonalne polega na sprawdzeniu „co" system robi, natomiast niefunkcjonalne „jak" system działa.

Testy funkcjonalne odpowiadają na pytanie jak system wykonuje funkcje, które ma za zadanie udostępniać włączając w to komendy użytkownika, operacje na danych, wyszukiwanie, procesy biznesowe i ekrany użytkownika. Testy funkcjonalne pokrywają zarówno funkcje dostępne dla użytkownika poprzez interfejs aplikacji jak i operacje typu back-end.

Testy funkcjonalne dotyczą funkcji lub innych cech oraz współdziałania z innymi systemami. Natomiast **testy niefunkcjonalne** to testy wymagane do zmierzenia właściwości systemów i oprogramowania, mogące zostać określone ilościowo na zmiennej skali, takich jak czasy odpowiedzi w testach wydajnościowych. Mogą się odnosić do modelu jakości oprogramowania.
W testach funkcjonalnych (testy czarnej skrzynki lub black box testing) tester nie ma dostępu do kodu testowanej aplikacji. Testy wykonywane są przez osobę, która nie tworzyła aplikacji w oparciu o dokumentację oraz założenia funkcjonalne. W zakres testów metodą czarnej skrzynki wchodzi badanie wartości brzegowych oraz definiowanie klas równoważności.
W tym rodzaju testów oprogramowanie jest sprawdzane pod kątem wymagań funkcjonalnych. Przypadki testowe pisane są pod kątem sprawdzenia czy aplikacja zachowuje się w oczekiwany sposób.
Testowanie funkcjonalne obejmuje:
• Testy jednostkowe
• Testy dymne / testy kondycji
• Testowanie integracyjne (testowanie zstępujące, testowanie wstępujące)

Patronat – INTIVE

Etap 1 QA

• Testy interfejsu i użyteczności
• Testy systemowe
• Testy regresji
• Testy Alfa i Beta
• Testy akceptacyjne użytkownika
• Testowanie metodami białej skrzynki oraz metodą czarnej skrzynki

Testowanie niefunkcjonalne obejmuje:
• Testowanie obciążeniowe i wydajnościowe
• Testowanie przeciążające i obciążenia
• Testowanie ergonomii
• Testowanie współpracy (międzyoperacyjności)
• Testowanie konwersji danych
• Testy bezpieczeństwa / Testy penetracyjne
• Testowanie instalacji
• Testy bezpieczeństwa (zabezpieczeń aplikacji, sieci)

2. **Co to są 'smoke testy' i testy regresji? Kiedy je stosujemy?**

Smoke test (test dymny)

Test dymny (smoke test) to podzbiór wszystkich zdefiniowanych/ zaplanowanych przypadków testowych, które pokrywają główne funkcjonalności modułu lub systemu, mający na celu potwierdzenie, że kluczowe funkcjonalności programu działają, bez zagłębiania się w szczegóły.
Smoke test wywodzi się z testowania sprzętu – podłączamy nowe urządzenie i sprawdzamy czy nie… wybuchnie. Jeśli pojawi się dym lub ogień wtedy wiemy, że coś poszło nie tak.
Nazwa określa po prostu szybkie testy które w jakimś stopniu potrafią sprawdzić czy aplikacja działa.
Somke testy mogą być wykonywane na podstawie istniejących przypadków testowych lub przy użyciu narzędzi do testów automatycznych.
Przeznaczeniem smoke testów jest "dotknięcie" każdej z części aplikacji bez zagłębiania się w logikę jej działania. **Chodzi o upewnienie się, czy najbardziej krytyczne funkcje działają.**

Smoke test mówi nam, czy program/system da się uruchomić, czy jego interfejsy są dostępne i czy reagują na działania użytkownika. Jeżeli smoke test nie powiedzie się nie ma powodu aby przechodzić do dalszych testów.

Cechy smoke test ("testu dymnego") to:
• **test pobieżny** (będzie to często test przeszukujący "wszerz", a nie "w głąb"; często będzie to test najbardziej typowej ścieżki czynności, którą może przebyć potencjalny użytkownik),
• **test zajmujący niewiele czasu** (ma szybko udzielić informacji koniecznych do podjęcia decyzji, co zrobimy dalej w ramach testowania),
• **test poszukujący bardzo wyraźnych problemów** (test zdany pomyślnie powinien wykluczyć zachodzenie ewidentnej awarii na swojej ścieżce),

• **test dopuszczający do kolejnego etapu prac** (zwłaszcza w kontekście zaangażowania w tym kolejnym etapie znaczących zasobów).

Kiedy stosujemy smoke test?
Smoke test przeprowadzany jest przez programistów tuż przed oddaniem wersji aplikacji lub przez testerów, przed zaakceptowaniem otrzymanej do testów aplikacji.
Mogą być zaprojektowane zarówno we wczesnych etapach wytwarzania oprogramowania, kiedy ustalane są najistotniejsze ścieżki przechodzenia przez aplikację - możemy na przykład wybrać najbardziej priorytetowe albo ryzykowne przypadki użycia (use case'y) i zaprojektować smoke test, który przejdzie przez te z najistotniejszych dla klienta, które pokrywają najwięcej obszarów aplikacji za jednym zamachem.
Nie wyklucza to możliwości tworzenia dodatkowych smoke testów później w procesie wytwarzania, gdy na przykład wydarzenia podczas projektu wskazują na prawdopodobieństwo zachodzenia "krytycznej" awarii. Bardzo typowe będzie dla zespołu testowego wykonywanie smoke testu, który konkretnie ma upewnić zespół, że, przykładowo, po ostatnim wdrożeniu najnowszej wersji systemu, ta wyjątkowo groźna awaria w znanym nam miejscu nie występuje.

Testy regresji (testowanie regresywne)
Jest to ponowne przetestowanie uprzednio testowanego programu po dokonaniu w nim modyfikacji, w celu upewnienia się, że w wyniku zmian nie powstały nowe defekty lub nie ujawniły się defekty w niezmienionej części oprogramowania. Czyli upewniamy się, że aplikacja działa po dokonaniu w niej modyfikacji, poprawieniu błędów lub po dodaniu nowej funkcjonalności.

Co dają testy regresji?:
• Wyszukanie błędów powstałych w wyniku zmian kodu/środowiska.
• Ujawnienie wcześniej nie odkrytych błędów.
• Jest to dobry kandydat do automatyzacji ze względu na swoją powtarzalność.
• Iteracyjne metodologie oraz krótkie cykle w których dostarczane są kolejne funkcjonalności powodują, że testy regresywne pozwalają upewnić się czy nowe funkcjonalności nie wpłynęły negatywnie na istniejące już i działające części systemu.

Kiedy stosujemy testowanie regresywne?
Testy regresyjne mogą zostać wykonane na wszystkich poziomach testowych i zajmują się funkcjonalnością, zdolnościami niefunkcjonalnymi i testowaniem strukturalnym.
Testy regresywne mogą być przeprowadzane dla kompletnego produktu lub jego części. Testy te są przeprowadzane po zmianie oprogramowania lub jego środowiska. Testy regresywne powinny być przeprowadzane po smoke testach lub testach typu sanity.

3. Co jest celem testowania?

Cele testowania:
• znajdowanie usterek
• nabieranie zaufania do poziomu jakości
• dostarczanie informacji potrzebnych do podejmowania decyzji
• zapobieganie defektom

Przykłady:
- W testowaniu wytwórczym (np. modułowym, integracyjnym lub systemowym) głównym celem może być wywołanie wielu awarii, aby zidentyfikować i poprawić usterki występujące w oprogramowaniu.
- W testach akceptacyjnych celem może być potwierdzenie, że system działa tak jak powinien oraz nabranie pewności, że spełnia wymagania.
- Celem może być też ocena jakości oprogramowania, by dostarczyć klientom informacji o ryzyku związanym z wydaniem systemu w danej chwili.
- Testowanie pielęgnacyjne sprawdza, czy nie wprowadzono nowych usterek podczas wykonywania zmian.
- W testowaniu produkcyjnym celem może być ocena atrybutów systemu np. niezawodności, dostępności.

4. Jak tester może się upewnić, że błąd został naprawiony?

Tester może się upewnić, że błąd został naprawiony wykonując retesty.

Retesty to testowanie polegające na uruchomieniu przypadków testowych, które podczas ostatniego uruchomienia wykryły błędy, w celu sprawdzenia poprawności naprawy. Późniejsze testy potwierdzające (retesty) wykonywane przez testera gwarantują, że poprawka rzeczywiście usunęła usterkę.

5. Testujesz aplikacje termometr która wykonuje pomiar temperatury. Co byś zrobił aby przetestować zachowanie aplikacji przy skrajnych wartościach -50C i 200C ?

Wartość skrajna to m.in. ekstremum, maksimum, minimum.

Przetestowałabym dla następujących wartości: -50 °C, 200 °C ewentualnie dla wartości brzegowych.

Możliwe by było zamokowanie i przesłanie zamokowanych danych, które symulują wartości jakie przekazuje urządzenie mierzące temperaturę.

Mockowanie to tzw. „udawanie”– dzięki temu możemy przetestować wybrany komponent, dostarczając mu ‚udawane’ zależności, przez co nie musimy np. tworzyć testowej bazy danych czy obawiać się, że jakieś dane zostaną zmienione podczas testów. Daje też wiele dodatkowych możliwości, które możemy wykorzystać w naszych testach .Mockowanie daje nam przewidywalny wynik działania konkretnego serwisu, niezależny od np. połączenia z innymi systemami, danych w bazie danych czy innych okoliczności. Dzięki temu unikamy sytuacji w której testy dają różne wyniki w zależności od tego, kiedy, jak i na jakim urządzeniu zostaną uruchomione.

Druga zaleta to możliwość rekonfiguracji – dodatkowa implementacja jakiegoś serwisu tylko na potrzeby testu jest ograniczona jeśli chodzi o skrajne przypadki. Mocki pozwalają nam zmieniać zachowanie mocków pomiędzy testami, czyniąc je wyjątkowo elastycznym i przydatnym narzędziem do testów.

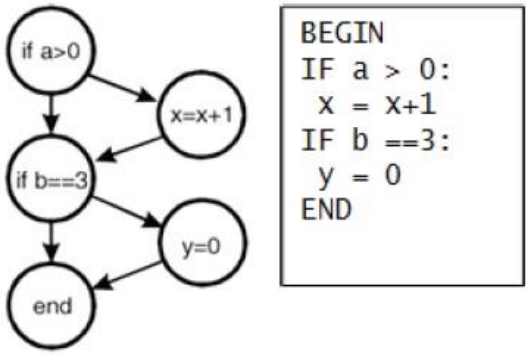
Etap 1 QA

Idea mockowania polega na tym, że test jednostkowy musi się wykonywać w izolacji od środowiska. Nie ma więc mowy o połączeniu się z bazą danych. Aby możliwe było wytestowanie usługi wykorzystującej bazę koniecznie jest dostarczenie zastępczych danych, aby móc wytestować logikę. Usługa pobierająca dane z bazy jest zastąpiona mockiem (którego zachowanie sami konfigurujemy, w zależności od potrzeb testu). Skupiamy się natomiast na wytestowaniu kodu, lecz zamiast zależności wstawiamy mocki. Dzięki temu uzyskujemy izolację testu od innych obiektów, co jest pożądane w unit testach.

Mocki umożliwiają też testowanie interakcji, czyli sprawdzenie czy obiekt testowany zachował zależność z konkretnymi parametrami, ile razy itp.

Można stworzyć mock, wypisać oczekiwania wobec niego (czyli jakie ma wywołać metody i co te metody mają zwrócić), a test powinien sprawdzić, czy te metody zostały wywołane, oraz jaką wartość zwróciły. Jeżeli metoda nie zostanie wywołana, choć oczekujemy tego - wówczas zostanie zwrócony błąd.

6. Ile przypadków testowych potrzeba, aby pokryć wszystkie możliwości?



Aby pokryć wszystkie możliwości potrzeba 4 przypadków testowych – aby pokryć wszystkie 4 ścieżki.

7. Dany jest input „wiek”, który przyjmuje wartości od 18 do 60. Twoim zadaniem jest przetestować go za pomocą techniki wartości brzegowych. Jakie wartości wpisujesz do inputu? Podaj wszystkie liczby, które wpisujesz.

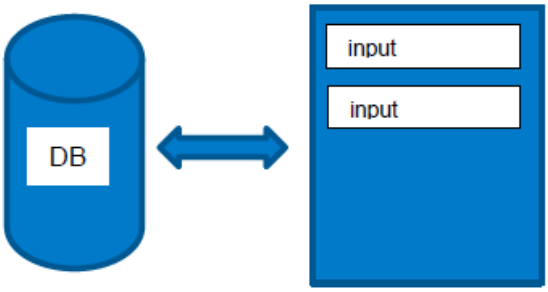
Wiek: 17,18,19,59,60,61

Wartość brzegowa to wartość znajdująca się wewnątrz, pomiędzy lub tuż przy granicy danej klasy równoważności. Istnieje duże prawdopodobieństwo, że oprogramowanie będzie się błędnie zachowywać dla wartości na krawędziach klas równoważności. Wartości brzegowe to na ogół minimum i maksimum (o ile istnieją) klasy równoważności.

Etap 1 QA

Ponieważ minimum i maximum klas równoważności to liczby 18 i 60, powinniśmy przetestować wiek dla wartości 17 oraz 61. W praktyce testujemy także dla wartości, znajdujących się pomiędzy wartościami 18 i 60 czyli dla 19 i 59.

8. Dołączasz do projektu w trakcie develepmentu aplikacji, do której nie ma dokumentacji. Schemat logowania do aplikacji wygląda następująco:



Jakie pytania zadasz analitykowi, zanim przystąpisz do testów logowania?

Pytania:

a) Na jaką platformę stworzona jest aplikacja?
b) Jaka przeglądarka obsługuje tą aplikację?
c) Czy transmisja jest szyfrowana i jaką metodą?
d) Czy zrobiono walidację oprogramowania?
e) Jakie warunki musi spełniać login?
f) Jakie warunki musi spełniać hasło?
g) Czy określono liczbę znaków w loginie i haśle?
h) Czy zastosowano znaki specjalne/ białe znaki?
i) f) Czy jest opcja automatycznego wylogowywana? Jeśli tak to po jakim czasie?
j) g) Czy jest możliwość łączenia się z bazą danych za pomocą różnych protokołów sieciowych (ftp, http itd.)?
k) Po jakiej stronie są przechowywane dane podczas pracy z tą aplikacją (klienta czy serwera)?
l) Czy stworzono już przypadek testowy dotyczący logowania?

9. Czym się różni metoda GET od POST?

Wszystkie transakcje WWW - a zatem także wysyłanie zawartości formularza - są realizowane przy użyciu protokołu HTTP. Protokół ten definiuje cztery metody przekazywania danych. W stosunku do formularzy zastosowanie znajdują dwie metody: GET oraz POST.

W metodzie GET dane są dołączone do adresu URL i przyjmują postać:

http://gdzies.w.sieci/kat/strona.php?imie=Jan&nazwisko=Nowak&plec=M&wiek=35

Najczęściej służy to do podania podstrony serwisu, którą chcemy obejrzeć, wersji językowej lub specjalnej wersji do wydruku. Metoda GET jest też często używana do przekazywania

identyfikatora sesji. Metoda GET powinna być stosowana tam, gdzie dane odwołanie powinno być adresowalne, to znaczy gdy w URL może być przechowywany pewien stan aplikacji, stały w czasie, do którego użytkownik mógłby chcieć wrócić (np. pozycja na mapie). Strony dostępne przez metodę GET powinny być bezpieczne dla osoby odwiedzającej (ang. *safe interactions*), tzn. nie powinny wywoływać wiążących dla niej skutków (np. zapisanie na listę dyskusyjną)

Natomiast w metodzie POST dane z formularza są dołączone na końcu zapytania HTTP (za wszystkimi nagłówkami). Metoda POST powinna być stosowana tam, gdzie dane odwołanie jest formą interakcji z użytkownikiem, niemożliwą do utrwalenia w formie adresu. Powinna być także stosowana tam, gdzie dana operacja może wywołać wiążące dla użytkownika skutki – np. zapisanie na listę dyskusyjną.

W zależności od użytej metody, dane pochodzące z formularza odbieramy na różne sposoby wewnątrz skryptu przetwarzającego formularz.

Podstawowymi różnicami między dwoma sposobami przesyłania danych z formularza są:

- metoda **GET** przesyła dane w adresie strony, po znaku *?*. Action wskazuje na adres strony, która się otworzy po wysłaniu formularza,
- metoda **POST** przesyła dane w sposób niezauważalny dla zwykłego użytkownika.

W języku PHP te dwie metody są używane w dwóch różnych miejscach:

- metoda **GET** służy do wysyłania danych na serwer,
- metoda **POST** służy do wstawiania danych do bazy, np. MySQL.

Metody GET używa się kiedy parametrów jest niewiele. Dzieje się tak ponieważ parametry przekazuje się za pomocą adresu URL (np. http://www.coś.pl/strona.php?parametr1=wartość1&parametr2=wartość2), którego długość jest ograniczona. Należy też pamiętać że parametry są widoczne w pasku adresu przeglądarki, więc tej metody nie należy używać jeśli przekazywane są np. hasła. Sam adres od parametrów oddzielany jest za pomocą znaku zapytania, a kolejne pary parametr=wartość za pomocą znaku ampersand („&”). Tą metodę można wykorzystać także przy przekazywaniu parametrów przez odnośnik.

Metoda „POST” do przekazywania parametrów wykorzystuje nagłówek zapytania – wystarczy wiedzieć, że metoda ta umożliwia przekazywanie dużo większych parametrów, a także że parametrów nie widać w pasku przeglądarki.

Zależnie od metody, zmienne trafiają do odpowiednich tablic asocjacyjnych. Dane przesłane metodą GET trafiają do tablicy $_GET (lub $HTTP_GET_VARS w wersjach starszych niż 4.1.0) a dane z metody POST to tablicy $_POST (lub $HTTP_POST_VARS). Tablice $_GET i $_POST są *superglobalne*. Oznacza to, że są widoczne w każdym miejscu kodu PHP bez konieczności użycia składni globals.

10. Czy HTTP jest protokołem zmiennostanowym?

Etap 1 QA

HTTP to skrót od Hypertext Transfer Protocol i jest to główny protokół używany współcześnie w przeglądarkach. **Jest to protokół bezstanowy**, tzn. ani serwer (ani klient) nie przechowuje informacji o tym, jakie były wcześniej zapytania pomiędzy określonym serwerem i klientem oraz nie posiada stanu wewnętrznego. Powoduje to, że każde zapytanie do serwera traktowane jest jako ‚nowe', z punktu widzenia serwera aplikacji niemożliwe do powiązania z informacjami np. o zalogowanym użytkowniku. Pozwala to znacznie zmniejszyć obciążenie serwera, jednak jest kłopotliwe w sytuacji, gdy np. trzeba zapamiętać konkretny stan dla użytkownika, który wcześniej łączył się już z serwerem.

11. Czym różni się LEFT JOIN od INNER JOIN?

Przy ich pomocy możemy łączyć ze sobą kilka tabel.

W przypadku serwera SQL Server kombinacja słów kluczowych INNER JOIN daje taki sam efekt, co instrukcja JOIN.

- **INNER JOIN** – łączenie tabel przy wyświetlaniu

W naszych tabelach przechowujemy różne dane, takie jak: pensje, adresy, czy stanowiska. Co w przypadku kiedy chcielibyśmy wyświetlić te dane w jednej tabeli? Możemy tego dokonać stosując operator **INNER JOIN** (złączenie wewnętrzne tabel) razem z poleceniem **SELECT**, które złączy wskazane przez nas tabele i kolumny. Polecenie będzie miało wtedy następującą składnię:

SELECT lista_kolumn FROM nazwa_tabeli INNER JOIN nazwa_tabeli_2 ON warunek_złączenia

Przykład:

Chcemy wyświetlić w jednej tabeli następujące dane: „id_pracownika", „imie", „nazwisko" oraz „adres_email". Trzy pierwsze dane znajdują się w tabeli „pracownik", zaś adres e-mail w tabeli „adresy". Wspólnym polem dla obu tabel jest „id_pracownika" i to ono posłuży nam do złączenia tabel następującym poleceniem:

SELECT pracownik.id_pracownika, imie, nazwisko, adres_email FROM pracownik INNER JOIN adresy ON pracownik.id_pracownika = adresy.id_pracownika

Tabela Złączenie tabel pracownik oraz adresy

id_pracownika	imie	nazwisko	adres_email
12008	Jan	Kowalski	jkowalski@msql.pl
22008	Adam	Nowakowski	anowak@msql.pl
32008	Kasia	Kowalska	*NULL*

W trzecim rzędzie pole adresu ma wartość **NULL**, ponieważ nie zawierało w bazie danych adresu e-mail. Warunek złączenia: pracownik.id_pracownika = adresy.id_pracownika składa się z nazwy_pierwszej_tabeli.wspolna_kolumna = nazwa_drugiej_tabeli.wspolna_kolumna.

Jeżeli chcielibyśmy do tej tabeli dołączyć jeszcze informacje na temat stanowiska pracownika przechowywane w polu „stanowisko" w tabeli „stanowiska" polecenie miałoby następującą formę:

SELECT pracownik.id_pracownika, imie, nazwisko, adres_email, stanowisko FROM pracownik INNER JOIN adresy ON pracownik.id_pracownika = adresy.id_pracownika INNER JOIN stanowiska ON pracownik.id_pracownika = stanowiska.id_pracownika

Tabela Złączenie tabel pracownik, adresy oraz stanowiska

id_pracownika	imie	nazwisko	adres_email	stanowisko
12008	Jan	Kowalski	jkowalski@msql.pl	przedstawiciel handlowy
22008	Adam	Nowakowski	anowak@msql.pl	księgowa
32008	Kasia	Kowalska	*NULL*	kasjer

W języku SQL mamy jeszcze następujące rodzaje złączeń:

- **LEFT JOIN** – lewostronne złączenie zewnętrzne, które zwraca wszystkie wiersze po lewej stronie klauzuli **JOIN**, zaś z tabeli po prawej stronie tylko wiersze dla których warunek złączenia jest prawdziwy. W przypadku kiedy wiersz z lewej tabeli nie ma odpowiadającego mu wiersza z prawej tabeli to odpowiednie pola zostają wypełniane wartością **NULL**.

Różnice pomiędzy LEFT JOIN a INNER JOIN:

- left join pobiera WSZYSTKO z lewej tabeli, a z prawej pobiera jedynie pasujące do klucza
- innerjoin pobiera jedynie pasujące do klucza

klucz w tym wypadku to jest to co znajduje się w "ON".

Jeśli zastosujemy instrukcję **LEFT JOIN** to w wynikach dostaniemy wszystko z lewej tabeli i dopasowane do nich elementy z prawej lub nulle. Jak natomiast zastosujemy instrukcję INNER JOIN to dostaniemy to samo, ale bez wierszy gdzie mieliśmy wartość null.

12. W jakim katalogu, standardowo Linux trzyma pliki konfiguracyjne:

a. /boot

b. /var

c. /etc

d. /cfg

Linux standardowo trzyma pliki konfiguracyjne w katalogu /etc.

13. Jak przetestowałbyś bashową komendę cp? (argumenty funkcji można pominąć)

cp to polecenie UNIXowe służące głównie do kopiowania plików, ale i katalogów. Pliki mogą być skopiowane w tym samym katalogu, innym katalogu, a nawet innym systemie plików czy dysku twardym. Jeśli plik jest kopiowany do tego samego katalogu, musi mieć inną nazwę, niż plik pierwotny. W pozostałych przypadkach nazwa może być taka sama lub zmieniona. Oryginalny plik pozostaje wtedy bez zmian. Komenda cp ma wiele wariantów, dwa najważniejsze zawarte są w GNU i POSIX.

Aby kopiować pliki lub katalogi za pomocą konsoli musimy użyć polecenia cp. Polecenia tego można użyć w trzech wariantach, tak więc poniżej pokazany został możliwy sposób użycia:

$ cp [opcje] [-T] źródło cel
$ cp [opcje] źródło katalog
$ cp [opcje] -t katalog źródło

W najbardziej podstawowym użyciu polega na podaniu dwóch argumentów, z czego pierwszy to plik (lub pliki), który ma zostać skopiowany, a drugi to miejsce do którego ma zostać skopiowany. Jeśli drugim argumentem jest nazwa pliku, to pierwszy plik zostanie skopiowany do drugiego (należy pamiętać, że podczas kopiowania nie wyskakuje żadne pytanie, czy na pewno chcemy plik podmienić/nadpisać, jeśli już o takiej samej nazwie istniał wcześniej).

Możliwe przypadki testowe:

Identyfikator: ID 1.1

Cel testu / Tytuł testu: Poprawne działanie polecenia kopiowanie plików: plik1 został skopiowany do plik2.

Warunki początkowe:
plik1 znajduje się w katalogu test.

Akcja / Kroki do wykonania:

1)	Wyświetlenie zawartości katalogu	**test ewa@eim:~/Desktop/test$ ls**
2)	Efekt działania polecenia	**kat1 kat2 kat3 plik1**
3)	Kopiowanie pliku plik1 do plik2	**ewa@eim:~/Desktop/test$ cp plik1 plik2**

Oczekiwany wynik testu:

1) W katalogu test znajduje się plik1 i plik2

ewa@eim:~/Desktop/test$ ls
kat1 kat2 kat3 plik1 plik2

Identyfikator: ID 1.2

Cel testu / Tytuł testu: Poprawne działanie kopiowania plików do katalogu.

Warunki początkowe:
plik1 znajduje się w katalogu test na pulpicie użytkownika ewa systemu eim.

Akcja / Kroki do wykonania:
1) Kopiowanie pliku plik1 do plik2 w kat1 **ewa@eim:~/Desktop/test$ cp plik1 plik2 kat1/**

Oczekiwany wynik testu:

1) W katalogu kat1 znajdują się plik1 i plik2

ewa@eim:~/Desktop/test$ ls kat1
kat1:
plik1 plik2

Identyfikator: ID 1.3

Cel testu / Tytuł testu: Poprawne działanie kopiowania plików do katalogu przez podanie ścieżki bezwzględnej.

Warunki początkowe:
plik1 znajduje się w katalogu test na pulpicie użytkownika ewa systemu eim.

Akcja / Kroki do wykonania:
1) Kopiowanie plików o początkowej nazwie plik do katalogu o ścieżce bezwzględnej

ewa@eim:~/Desktop/test$ cp plik*/home/ewa/Desktop/test/kat2

Oczekiwany wynik testu:

1) W katalogu kat2 znajdują się skopiowane pliki.

ewa@eim:~/Desktop/test$ ls kat2
kat2:
plik1 plik2

Identyfikator: ID 1.4

Etap 1 QA

Cel testu / Tytuł testu: Poprawne działanie kopiowania plików do katalogu przez podanie ścieżki względnej.

Warunki początkowe:
plik1 znajduje się w katalogu test na pulpicie użytkownika ewa systemu eim.

Akcja / Kroki do wykonania:
1) Kopiowanie plików o początkowej nazwie plik do katalogu o ścieżce względnej

ewa@eim:~/Desktop/test$ cp plik* ~/Desktop/test/kat3

Oczekiwany wynik testu:

1) W katalogu kat3 znajdują się skopiowane pliki.

ewa@eim:~/Desktop/test$ ls kat3
kat3:
plik1 plik2

Identyfikator: ID 1.5

Cel testu / Tytuł testu: Poprawne działanie kopiowania katalogu z zawartością: cp -r

Warunki początkowe:
kat1 znajduje się w katalogu test.

Akcja / Kroki do wykonania:
1) Kopiowanie zawartości katalogu kat1 do katalogu kat2.
   ewa@eim:~/Desktop/test$ cp -r kat1/ kat2/

Oczekiwany wynik testu:

1) W katalogu kat2 znajduje się zawartość kat1.
   ewa@eim:~/Desktop/test$ ls kat2
   kat2:
   kat1 plik1 plik2

Identyfikator: ID 1.6

Cel testu / Tytuł testu: Nadpisanie pliku podczas kopiowania z powiadomieniem.
Warunki początkowe:
plik1 i plik2 znajdują się w katalogu test.

Akcja / Kroki do wykonania:
1) Kopiowanie plik1 do plik2 z powiadomieniem o próbie zastąpienia pliku.

Patronat – INTIVE

Etap 1 QA

ewa@eim:~/Desktop/test$ cp -i plik1 plik2
cp: overwrite ‘plik2’? y

Oczekiwany wynik testu:

1) Nadpisanie pliku podczas kopiowania.

Można też przetestować bashową komendę cp tworząc przypadki testowe:

1) Kopiowanie plików na inny dysk
2) Kopiowanie z podaniem błędnych parametrów czyli podajemy np. samą nazwę pliku bez katalogu.
3) Kopiowanie samego katalogu bez pliku
4) Błędne podanie nazwy pliku
5) Kopiowanie plików na nieistniejący dysk
   Itd.