# Sylabus dla modułu specjalistycznego

# Poziom podstawowy - Tester Wydajności

wersja 2018

---

International Software Testing Qualifications Board®


---



---

Opracowane przez

American Software Testing Qualifications Board®

i

German Testing Board®

---


Informacja o prawach autorskich



Grupa Robocza ds. Testów Wydajnościowych:

Graham Bath
Rex Black
Alexander Podelko
Andrew Pollner
Randy Rice



Tłumaczenie z języka angielskiego oraz udział w przeglądach wersji 2018: Damian Szczurek.

Przegląd końcowy:
Grzegorz Holak, Jan Sabak, Monika Petri-Starego

Redakcja i korekta: Monika Petri-Starego

# Historia zmian

| Wersja | Data | Uwagi |
|---|---|---|
| Version 2018 | 24.02.2021 | Przegląd edytorski |
| Version 2018 | 9.12.2018 r. | Wydanie ISTQB® GA |
| V2018 b3 | 25.08.2018 r. | Komentarze do recenzji beta włączone dla wersji wydania |
| V2018 b2 | 17.05. 2018 r. | Wydanie beta dla ISTQB® |
| V2018 b1 | 1.03.2018 r. | Kandydat beta do ISTQB® |
| V2017v4 | 23.01.2018 r. | Przegląd słownika |
| V2017v3 | 15.12.2018 r. | Edycja techniczna |
| V2017v2 | 15.12.2017 r. | Aktualizacje alfa |
| V2017v1 | 27.11.2017 r. | Do przeglądu alfa |
| Alpha Review V10 | 28.06.2017 r. | Do przeglądu alfa |
| Alpha V09 | 16.04.2017 r. | Wersja pre-alfa |
| Alpha V08 | 12.02.2017 r. | Wersja pre-alfa |
| Alpha V07 | 31.12.2016 r. | Dodane komentarze autora |
| Alpha V06 | 23.12.2016 r. | Zmiany Roz. 4<br>Zmiana numeracji i dostosowanie CN zgodnie z ustaleniami na NYC |
| Alpha V05 | 18.12.2016 r. | Po spotkaniu NYC |
| Alpha V04 | 13.12.2016 r. | Wersja na spotkanie NYC |

# Spis treści

# Podziękowania

Ten dokument został opracowany przez American Software Testing Qualifications Board (ASTQB®) i German Testing Board (GTB®):

Graham Bath (GTB®, współprzewodniczący Grupy Roboczej)
Rex Black
Alexander Podelko (CMG®)
Andrew Pollner (ASTQB®, współprzewodniczący Grupy Roboczej)
Randy Rice.

Główny zespół dziękuje zespołowi recenzentów za sugestie i wkład. ASTQB® pragnie podziękować Computer Measurement Group (CMG) za ich wkład w opracowanie tego sylabusa.

W recenzowaniu, komentowaniu lub głosowaniu na ten plan nauczania lub jego wcześniejsze wersje brały udział następujące osoby:

Dani Almog
Sebastian Chece
Todd DeCapua
Wim Decoutere
Frans Dijkman
Jiangru Fu
Matthias Hamburg
Ágota Horváth Meile
Mieke Jungebload
Beata Karpińska
Gábor Ladányi
Kai Lepler
Ine Lutterman
Marek Majernik
Stefan Massonet
Judy McKay
Gary Mogyorodi
Joern Muenzel
Petr Neugebauer
Ingvar Nordström
Meile Posthuma
Michaël Pilaeten
Filip Rechtoris
Adam Roman
Dirk Schweier
Marcus Seyfert
Péter Sótér
Michael Stahl
Jan Stiller
Federico Toledo
Andrea Szabó
Yaron Tsubery
Stephanie Ulrich
Mohit Verma
Armin Wachter
Huaiwen Yang
Ting Ya.

Dokument ten został oficjalnie opublikowany przez ISTQB® 9 grudnia 2018 r.

# 0. Wprowadzenie do sylabusa (planu nauczania)

## 0.1. Cel dokumentu

Niniejszy sylabus (plan nauczania) stanowi podstawę do certyfikacji ISTQB® Tester Wydajności Poziomu Podstawowego.
ASTQB® i GTB® udostępniają niniejszy sylabus:

1. Radom Krajowym celem tłumaczenia na ich języki lokalne i dokonania akredytacji dostawców szkoleń. Rady Krajowe mogą dostosowywać sylabus do potrzeb danego języka i dodawać odwołania do literatury w celu dostosowania do publikacji lokalnych,
2. organom certyfikującym 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 informatycznych w celu rozwijania zawodu testera oprogramowania i systemów informatycznych oraz opracowywania książek i artykułów.

ASTQB® i GTB® mogą również zezwolić innym podmiotom na korzystanie z niniejszego sylabusa do innych celów pod warunkiem wystąpienia przez te podmioty o stosowną pisemną zgodę.

## 0.2. Certyfikacja na poziomie podstawowym - Tester Wydajności

Certyfikacja na poziomie podstawowym jest skierowana do każdego, kto zajmuje się testowaniem oprogramowania i chce poszerzyć swoją wiedzę na temat testowania wydajnościowego lub do kogokolwiek, kto chce rozpocząć karierę specjalistyczną w dziedzinie testów wydajnościowych. Certyfikacja jest również skierowana do każdego, kto jest zaangażowany w inżynierię wydajności i chce lepiej zrozumieć zagadnienia testów wydajnościowych.

Sylabus uwzględnia następujące, główne aspekty testowania wydajnościowego:

- aspekty techniczne,
- aspekty dotyczące metod,
- aspekty organizacyjne.

Informacje na temat testów wydajnościowych opisanych w sylabusie ISTQB® Poziom Zaawansowany - Techniczny Analityk Testów [ISTQB_ALTTA_SYL] są zgodne z informacjami zawartymi w niniejszym sylabusie.

## 0.3. Rezultaty biznesowe

Ten paragraf zawiera listę rezultatów biznesowych oczekiwanych od kandydata, który uzyskał certyfikat z obszaru testów wydajnościowych na poziomie podstawowym.

| | |
|---|---|
| PTFL-1 | Zrozumienie podstawowych koncepcji dotyczących zagadnień wydajności i testów wydajnościowych. |
| PTFL-2 | Zdefiniowanie ryzyk, celów i wymagań dotyczących wydajności tak, aby spełnić potrzeby i oczekiwania interesariuszy. |
| PTFL-3 | Zrozumienie metryk wydajności i tego, jak je gromadzić. |
| PTFL-4 | Opracowanie planu testów wydajnościowych, aby osiągnąć zakładane cele i spełnić wymagania. |
| PTFL-5 | Koncepcyjne projektowanie, zaimplementowanie i wykonanie podstawowego testu wydajnościowego. |
| PTFL-6 | Analiza rezultatów testów wydajnościowych i przedstawienie ich następstw różnym interesariuszom. |
| PTFL-7 | Wyjaśnienie procesu, zasadności, rezultatów i konsekwencji testów wydajnościowych różnym interesariuszom. |
| PTFL-8 | Zrozumienie kategorii i sposobów użycia narzędzi do testów wydajnościowych oraz zasad ich doboru. |
| PTFL-9 | Określenie, w jaki sposób czynności testowania wydajnościowego są dostosowane do cyklu życia oprogramowania. |

## 0.4. Cele nauczania objęte egzaminem

Cele nauczania wspierają rezultaty biznesowe i są wykorzystywane do tworzenia egzaminów służących uzyskaniu certyfikatu ISTQB® Tester Wydajności - Poziom Podstawowy. Cele nauczania są przypisane do poziomu poznawczego wiedzy (poziom K).

Poziom K, czyli poziom poznawczy, jest wykorzystywany do klasyfikacji celów nauczania zgodnie ze zmienioną taksonomią Blooma [Anderson01]. ISTQB® wykorzystuje tę taksonomię do projektowania swoich programów nauczania.

Ten sylabus obejmuje cztery różne poziomy K (od K1 do K4):

| Poziom K | Znaczenie | Opis |
|---|---|---|
| 1 | Zapamiętać | Kandydat powinien zapamiętać lub rozpoznać termin lub pojęcie. |
| 2 | Zrozumieć | Kandydat powinien wskazać wyjaśnienie dla stwierdzenia związanego z tematem pytania. |
| 3 | Zastosować | Kandydat powinien wskazać właściwe zastosowanie koncepcji lub techniki i zastosować ją w danym kontekście. |
| 4 | Przeanalizować | Kandydat potrafi podzielić informacje odnoszące się do procedury lub techniki na jej części składowe w celu lepszego zrozumienia i odróżnienia faktów od wniosków. |

Ogólnie rzecz biorąc, wszystkie części tego sylabusa (planu nauczania) mogą być przedmiotem egzaminu na poziomie K1. Oznacza to, że kandydat rozpozna, zapamięta i przypomni sobie dany termin lub pojęcie. Cele nauczania na poziomie K2, K3 i K4 są przedstawione na początku odpowiedniego rozdziału.

## 0.5. Rekomendowany czas trwania szkolenia

W niniejszym sylabusie określono minimalny czas szkolenia dla każdego celu nauczania. W nagłówku rozdziału podano całkowity czas trwania każdego rozdziału.

Dostawcy szkoleń powinni zwrócić uwagę na to, że inne sylabusy (plany nauczania) ISTQB® stosują podejście „standardowego czasu", które przypisuje określony czas trwania zgodnie z poziomem K (K-Level). Sylabus dla testowania wydajnościowego nie stosuje się ściśle do tego schematu. W rezultacie, dostawcy szkoleń otrzymują bardziej elastyczne i realistyczne wskazanie minimalnego czasu szkolenia.

## 0.6. Wymagania wstępne

Przed przystąpieniem do egzaminu ISTQB® Tester Wydajności - Poziom Podstawowy należy uzyskać certyfikat ISTQB® Tester Oprogramowania - Poziom Podstawowy.

## 0.7. Źródła informacji

Terminy użyte w sylabusie są zdefiniowane w dokumencie: ISTQB® Słownik terminów testowych ISTQB® [ISTQB_GLOSSARY].
Polska wersja Słownika jest dostępna na stronie https://glossary.istqb.org

Rozdział 6. zawiera listę zalecanych książek i artykułów na temat testowania wydajnościowego.

# 1. Wprowadzenie i podstawy — 60 min.

**Słowa kluczowe**

testowanie przepustowości, testowanie współbieżności, efektywność, testowanie wytrzymałościowe, generowanie obciążenia, testowanie obciążenia, testowanie wydajnościowe, testowanie skalowalności, testowanie skokowe, testowanie przeciążające

**Cele nauczania – wprowadzenie i podstawy**

**1.1. Zasady i koncepcje**

PTFL-1.1.1. (K2) Kandydat rozumie zasady testowania wydajnościowego.

**1.2. Typy testów wydajnościowych**

PTFL-1.2.1. (K2) Kandydat rozumie różnicę między typami testów wydajnościowych.

**1.3. Rodzaje testów w testowaniu wydajnościowym**

PTFL-1.3.1. (K1) Kandydat potrafi przywołać typy testów w procesie testowania wydajnościowego.

**1.4. Koncepcja generowania obciążenia**

PTFL-1.4.1. (K2) Kandydat rozumie koncepcję generowania obciążenia.

**1.5. Typowe rodzaje awarii w testach wydajnościowych i ich przyczyny**

PTFL-1.5.1. (K2) Kandydat potrafi podać przykłady typowych rodzajów awarii występujących podczas testów wydajnościowych oraz ich przyczyny.

## 1.1. Zasady testowania wydajnościowego

Efektywność wydajności (lub po prostu „wydajność”) jest istotną częścią zapewniania “pozytywnego wrażenia” użytkownikom podczas korzystania z aplikacji na różnych platformach stacjonarnych i mobilnych. Testowanie wydajnościowe odgrywa kluczową rolę w ustalaniu akceptowalnych poziomów jakości dla użytkownika końcowego i często jest ściśle powiązane z innymi dziedzinami, takimi jak inżynieria użyteczności i inżynieria wydajności.

Ponadto ocena przydatności funkcjonalnej, użyteczności i innych cech jakościowych w warunkach obciążenia, takich jak podczas wykonywania testu wydajnościowego, może ujawnić problemy specyficzne dla obciążenia, które mają wpływ na te właściwości.

Testowanie wydajnościowe nie ogranicza się do domeny internetowej, na której koncentruje się użytkownik końcowy. Ma również znaczenie dla aplikacji z innych obszarów, z różnymi architekturami systemu takimi jak: klasyczne systemy klient-serwer, systemy rozproszone i wbudowane (embedded). Z technicznego punktu widzenia efektywność wydajności jest sklasyfikowana w standardzie ISO 25010 [ISO25000] jako niefunkcjonalna

charakterystyka jakościowa z trzema podcharakterystykami jakościowymi opisanymi poniżej. Właściwa koncentracja i ustalenie priorytetów zależy od oceny ryzyka i potrzeb różnych interesariuszy. Analiza wyników testów umożliwi identyfikację innych obszarów ryzyka, którymi należy się zająć.

**Zachowanie w czasie:** Na ogół ocena zachowania w czasie jest najbardziej powszechnym celem testowania wydajnościowego. Ten aspekt testowania wydajnościowego bada zdolność komponentu lub systemu do reagowania na dane wejściowe użytkownika lub systemu w określonym czasie i w określonych warunkach. Pomiary zachowania w czasie mogą się różnić od całkowitego czasu potrzebnego systemowi na reakcję na dane wejściowe użytkownika w stosunku do liczby cykli CPU wymaganych przez komponent oprogramowania do wykonania określonego zadania.

**Wykorzystanie zasobów:** Jeśli dostępność zasobów systemowych zostanie zidentyfikowana jako ryzyko, wykorzystanie tych zasobów (np. przydzielenie ograniczonej pamięci RAM), może zostać zbadane poprzez przeprowadzenie określonych testów wydajnościowych.

**Przepustowość:** Jeżeli problemy związane z zachowaniem systemu przy wymaganych limitach wydajności systemu (np. liczba użytkowników lub wolumeny danych) zostaną zidentyfikowane jako ryzyko, można przeprowadzić testy wydajnościowe w celu oceny adekwatności architektury systemu.

Testowanie wydajnościowe często przybiera formę eksperymentowania, co umożliwia pomiar i analizę określonych parametrów systemu. Można je przeprowadzać iteracyjnie w celu wsparcia analizy, projektowania i wdrażania systemu, aby umożliwić podejmowanie decyzji architektonicznych i pomóc w kształtowaniu oczekiwań interesariuszy.

Poniższe zasady testowania wydajnościowego są szczególnie istotne.

- Testy muszą być dostosowane do określonych oczekiwań różnych grup interesariuszy, w szczególności użytkowników, projektantów systemów i personelu operacyjnego.
- Testy muszą być możliwe do odtworzenia (reprodukowalne). Należy uzyskać statystycznie identyczne wyniki (w określonej tolerancji) powtarzając testy na niezmienionym systemie.
- Testy muszą dawać wyniki, które są zarówno zrozumiałe, jak i łatwe do porównania z oczekiwaniami interesariuszy.
- Testy można przeprowadzać, o ile pozwalają na to zasoby, na kompletnych lub częściowych systemach lub środowiskach testowych, które są reprezentatywne dla systemu produkcyjnego.
- Testy muszą mieścić się w budżecie i być i możliwe do wykonania w ramach czasowych określonych w projekcie.

Książki [Molyneaux09] i [Microsoft07] zapewniają solidne podstawy w kwestii zasad i praktycznych aspektów testowania wydajnościowego.

Wszystkie trzy z powyższych podcharakterystyk jakościowych będą miały wpływ na zdolność testowanego systemu (SUT - ang. *System Under Test*) do skalowania.

## 1.2. Typy testów wydajnościowych

Można zdefiniować różne typy testów wydajnościowych. Każdy z nich może mieć zastosowanie do danego projektu, w zależności od celów testu.

**Testowanie wydajnościowe**

Testowanie wydajnościowe to termin ogólny obejmujący wszelkiego rodzaju testy skoncentrowane na wydajności (responsywności) systemu lub komponentu przy różnym obciążeniu.

**Testowanie obciążenia**

Testowanie obciążenia koncentruje się na ocenie zdolności systemu do obsługi rosnących poziomów przewidywanych, realistycznych obciążeń, wynikających z żądań transakcji generowanych przez kontrolowaną liczbę współbieżnych użytkowników lub procesów.

**Testowanie przeciążające**

Testy przeciążające koncentrują się na ocenie zdolności systemu lub komponentu do obsługi szczytowych obciążeń, które są równe lub przekraczają granice przewidywanych lub określonych obciążeń. Testy przeciążające są również wykorzystywane do oceny zdolności systemu do radzenia sobie z ograniczoną dostępnością zasobów, takich jak dostępna moc obliczeniowa, dostępna przepustowość i pamięć.

**Testowanie skalowalności**

Testowanie skalowalności koncentruje się na ocenie zdolności systemu do spełnienia przyszłych wymagań dotyczących wydajności, które mogą wykraczać poza obecnie wymagane. Celem tych testów jest określenie zdolności systemu do rozwoju (np. przy większej liczbie użytkowników, większej ilości przechowywanych danych) bez naruszania obecnie określonych wymagań dotyczących wydajności lub awaryjności. Gdy znane są granice skalowalności, można ustawić wartości graniczne i monitorować je w środowisku produkcyjnym, aby zapewnić ostrzeganie o problemach, które mogą się pojawić. Ponadto środowisko produkcyjne można przystosowywać za pomocą odpowiedniej ilości sprzętu.

**Testowanie skokowe**

Testowanie skokowe skupia się na ocenie zdolności systemu do prawidłowego reagowania na nagłe przypływy obciążeń skokowych, a następnie powrotu do stanu stabilnego.

**Testowanie wytrzymałościowe**

Testowanie wytrzymałościowe koncentruje się na ocenie stabilności systemu w przedziale czasowym specyficznym dla kontekstu operacyjnego testowanego systemu. Ten typ testowania umożliwia sprawdzenie, czy nie występują problemy z pojemnością zasobów (np. wycieki pamięci, połączenia z bazą danych, pule wątków), które mogą ostatecznie obniżyć wydajność i/lub spowodować awarie w punktach krytycznych.

**Testowanie współbieżności**

Testowanie współbieżności koncentruje się ocenie wpływu sytuacji, w których określone akcje występują jednocześnie (np. gdy w tym samym czasie loguje się duża liczba użytkowników). Problemy ze współbieżnością są niezwykle trudne do znalezienia i odtworzenia, szczególnie, gdy problem występuje w środowisku, nad którym testerzy mają małą kontrolę lub też nie ma jej wcale, na przykład w środowisku produkcyjnym.

**Testowanie przepustowości**

Testowanie przepustowości umożliwia określenie, ilu użytkowników i/lub transakcji dany system będzie w stanie obsłużyć i nadal spełniać określone cele wydajnościowe. Cele te można również określić w odniesieniu do ilości danych wynikających z transakcji.

## 1.3. Rodzaje testów w procesie testowania wydajnościowego

Główne rodzaje testów wykorzystywane w procesie testowania wydajnościowego obejmują testy statyczne i testy dynamiczne.

### 1.3.1. Testowanie statyczne

Czynności testowania statycznego są często ważniejsze dla testowania wydajnościowego niż dla testowania przydatności funkcjonalnej. Dzieje się tak, ponieważ wiele krytycznych defektów wydajności jest wprowadzanych do architektury i projektu systemu. Wady te mogą być spowodowane nieporozumieniami lub brakiem wiedzy projektantów i architektów. Wady te mogą być również wprowadzone, ponieważ wymagania nie uwzględniły odpowiedniego czasu odpowiedzi, przepustowości lub celów wykorzystania zasobów, oczekiwanego obciążenia i użytkowania systemu lub ograniczeń.

Do czynności testowania statycznego dla testów wydajnościowych zaliczamy:

- przeglądy wymagań ze szczególnym uwzględnieniem aspektów wydajności i ryzyka,

- przeglądy schematów baz danych, diagramów relacji encji, metadanych, procedur składowanych i zapytań,
- przeglądy systemu i architektury sieci,
- przeglądy krytycznych segmentów kodu systemu (np. złożone algorytmy).

### 1.3.2. Testowanie dynamiczne

W miarę budowania systemu dynamiczne testy wydajnościowe powinny rozpocząć się tak szybko, jak to możliwe. Dynamiczne testy wydajnościowe można wykorzystać:

- podczas testów jednostkowych, w tym z wykorzystaniem informacji o profilowaniu, w celu określenia potencjalnych wąskich gardeł i analizy dynamicznej w celu oceny wykorzystania zasobów,
- podczas testowania integracji komponentów, w kluczowych przypadkach użycia i przepływach pracy, zwłaszcza podczas integracji różnych funkcji przypadków użycia lub integracji ze strukturą „szkieletową" przepływu pracy,
- podczas testowania całych procesów biznesowych w ramach testowania systemowego, w różnych warunkach obciążenia,
- podczas testowania integracji systemów, szczególnie w przypadku przepływów danych i przepływów pracy między kluczowymi interfejsami między systemami. W testowaniu integracji systemów nierzadko „użytkownikiem" jest inny system lub maszyna (np. dane wejściowe z wejść czujników i innych systemów),
- podczas testowania akceptacyjnego, aby zbudować zaufanie użytkownika, klienta i operatora co do prawidłowego działania systemu i dostosować system w warunkach rzeczywistych (ale generalnie nie w celu znalezienia defektów wydajności w systemie).

Na wyższych poziomach testów, takich jak testy systemowe i testy integracji systemów, użycie realistycznych środowisk, danych i obciążeń ma kluczowe znaczenie dla dokładnych wyników (patrz Rozdział 4.). W zwinnych i innych iteracyjno-przyrostowych cyklach życia oprogramowania, w celu ograniczenia ryzyka związanego z wydajnością, zespoły powinny uwzględniać statyczne i dynamiczne testy wydajnościowe we wczesnych iteracjach, zamiast czekać na końcowe etapy projektu.

Jeśli częścią systemu jest niestandardowy lub nowy sprzęt, wczesne dynamiczne testy wydajnościowe można przeprowadzić przy użyciu symulatorów. Jednak dobrą praktyką jest jak najszybsze rozpoczęcie testów na rzeczywistym sprzęcie, ponieważ symulatory często nie wychwytują odpowiednio ograniczeń zasobów i zachowań związanych z wydajnością.

## 1.4. Koncepcja generowania obciążenia

Aby przeprowadzić różne rodzaje testów wydajnościowych opisanych w podrozdziale 1.2., należy zamodelować, wygenerować i przesłać reprezentatywne obciążenia systemu do testowanego systemu. Obciążenia są porównywalne z danymi wejściowymi używanymi w przypadku testów funkcjonalnych, ale różnią się w następujący sposób:

- testy obciążenia muszą reprezentować wiele danych wejściowych użytkownika, a nie tylko pojedynczy zestaw danych,
- testy wydajnościowe mogą wymagać specjalnego sprzętu i narzędzi do generowania obciążenia,
- generowanie testów wydajnościowych jest uzależnione od braku jakichkolwiek defektów funkcjonalnych w testowanym systemie, które mogą mieć wpływ na wykonanie testu.

Wydajne i niezawodne generowanie określonego obciążenia jest kluczowym czynnikiem sukcesu podczas przeprowadzania testów wydajnościowych. Istnieją różne opcje generowania obciążenia.

**Generowanie obciążenia za pośrednictwem interfejsu użytkownika**

Może to być odpowiednie podejście, jeśli reprezentowana ma być tylko niewielka liczba użytkowników i jeśli dostępna jest wymagana liczba oprogramowania klienckiego, z którego można wprowadzić wymagane dane wejściowe. Podejście to można również zastosować w połączeniu z narzędziami do wykonywania testów funkcjonalnych, ale może szybko stać się niepraktyczne wraz ze wzrostem liczby symulowanych użytkowników. Stabilność interfejsu użytkownika (UI) również stanowi krytyczną zależność. Częste zmiany mogą wpływać na powtarzalność testów wydajnościowych i mogą znacząco wpływać na koszty utrzymania. Testowanie za pośrednictwem interfejsu użytkownika może być najbardziej reprezentatywnym podejściem do testów kompleksowych (*end-to-end*).

**Generowanie obciążenia za pomocą testowania w tłumie (ang. *crowd testing*)**

Takie podejście zależy od dostępności dużej liczby testerów, którzy będą reprezentować rzeczywistych użytkowników. W testowaniu w tłumie testerzy są zorganizowani w taki sposób, aby można było wygenerować pożądane obciążenie. Może to być odpowiednia metoda testowania aplikacji, do których jest dostęp z dowolnego miejsca na świecie (np. aplikacje webowe) i może wymagać od użytkowników generowania obciążenia z wielu różnych typów urządzeń i w różnych konfiguracjach. Takie podejście może zaangażować do testów bardzo dużą liczbę użytkowników, jednak generowane obciążenie może nie być tak powtarzalne i precyzyjne jak w innych metodach, a jego zorganizowanie jest bardziej skomplikowane.

**Generowanie obciążenia za pomocą interfejsu programistycznego aplikacji (API)**

To podejście jest podobne do użycia interfejsu użytkownika do wprowadzania danych, ale wykorzystuje interfejs API aplikacji zamiast interfejsu użytkownika do symulacji interakcji użytkownika z testowanym systemem. Podejście to jest zatem mniej wrażliwe na zmiany (np. opóźnienia) w interfejsie użytkownika i umożliwia przetwarzanie transakcji w taki sam sposób, jak gdyby były wprowadzane bezpośrednio przez użytkownika za pośrednictwem interfejsu użytkownika. Można tworzyć dedykowane skrypty, które wielokrotnie wywołują określone procedury API i umożliwiają symulację większej liczby użytkowników w porównaniu z wykorzystaniem danych wejściowych interfejsu użytkownika.

**Generowanie obciążenia przy użyciu przechwyconych protokołów komunikacyjnych**

Podejście to obejmuje przechwytywanie interakcji użytkownika z testowanym systemem na poziomie protokołu komunikacyjnego, a następnie odtwarzanie tych skryptów w celu symulacji potencjalnie bardzo dużej liczby użytkowników w powtarzalny i niezawodny sposób. To podejście oparte na narzędziach opisano w sekcjach 4.2.6. i 4.2.7.

## 1.5. Typowe rodzaje awarii w testach wydajnościowych i ich przyczyny

Chociaż z pewnością istnieje wiele różnych rodzajów awarii związanych z wydajnością, które można wykryć podczas testów dynamicznych, poniżej przedstawiono kilka przykładów typowych awarii (w tym awarii systemu) wraz z opisem ich typowych przyczyn.

**Zbyt wolne odpowiedzi przy każdym poziomie obciążenia**
W niektórych przypadkach czas odpowiedzi jest niedopuszczalny, niezależnie od poziomu obciążenia. Może to być spowodowane podstawowymi problemami z wydajnością, w tym między innymi błędnym projektem lub implementacją bazy danych, opóźnieniem sieci i innymi obciążeniami w tle. Takie problemy można zidentyfikować podczas testów funkcjonalnych i użyteczności, a nie tylko testów wydajnościowych, więc analitycy testów (testerzy) powinni zwracać na nie uwagę i zgłaszać je.

**Zbyt wolne odpowiedzi przy poziomie obciążenia wzrastającym od średniego do dużego**
W niektórych przypadkach czasy odpowiedzi pogarszają się w stopniu niedopuszczalnym, przy obciążeniu wzrastającym od poziomu średniego do dużego, nawet jeśli takie obciążenia zawierają się w całkowicie normalnych, oczekiwanych, dozwolonych zakresach. Podstawowe defekty obejmują przesycenie jednego lub więcej zasobów oraz inne obciążenia w tle.

**Pogarszające się czasy odpowiedzi**
W niektórych przypadkach czasy odpowiedzi pogarszają się stopniowo lub poważnie w czasie. Podstawowymi przyczynami są wycieki pamięci, fragmentacja dysku, zwiększające się z czasem obciążenie sieci, powiększanie repozytorium plików i nieoczekiwany wzrost bazy danych.

**Nieadekwatna lub nierzetelna obsługa błędów przy dużym lub nadmiernym obciążeniu**
W niektórych przypadkach czas odpowiedzi jest akceptowalny, ale obsługa błędów pogarsza się przy wysokim i poza wyspecyfikowaną granicą obciążeniu. Podstawowe defekty obejmują niewystarczające pule zasobów, niedostatecznych rozmiarów (za krótkie, za małe) kolejki i stosy oraz zbyt niskie ustawienia limitu czasu.

Konkretne przykłady ogólnych typów awarii wymienionych powyżej obejmują:

- Aplikacja webowa, która dostarcza informacji o usługach firmy, nie odpowiada na żądania użytkowników w ciągu siedmiu sekund (ogólna zasada branżowa). Nie można osiągnąć efektywnego działania systemu w określonych warunkach obciążenia.
- System ulega awarii lub nie jest w stanie odpowiedzieć na dane wejściowe użytkownika, gdy jest poddawany nagłej dużej liczbie żądań użytkowników (np. sprzedaż biletów na duże wydarzenie sportowe). Zdolność systemu do obsługi takiej liczby użytkowników jest niewystarczająca.
- Reakcja systemu ulega znacznemu pogorszeniu, gdy użytkownicy przesyłają żądania dotyczące dużych ilości danych (np. duży i ważny raport jest publikowany na stronie internetowej do pobrania). Zdolność systemu do obsługi generowanych ilości danych jest niewystarczająca.
- Przetwarzanie wsadowe nie może się zakończyć, zanim będzie potrzebne przetwarzanie online. Czas realizacji procesów wsadowych jest niewystarczający w dozwolonym okresie.
- Systemowi czasu rzeczywistego zabraknie pamięci RAM, gdy równoległe procesy generują duże zapotrzebowanie na pamięć dynamiczną, której nie można zwolnić na czas. Pamięć RAM nie jest odpowiednio zwymiarowana lub żądania dotyczące pamięci RAM nie mają odpowiedniego priorytetu.
- Składnik A systemu czasu rzeczywistego, który dostarcza dane wejściowe do składnika B systemu czasu rzeczywistego, nie jest w stanie obliczyć aktualizacji z wymaganą szybkością. Cały system nie reaguje na czas i może zawieść. Moduły kodu w komponencie A muszą zostać ocenione i zmodyfikowane („profilowanie wydajności”), aby zapewnić osiągnięcie wymaganych szybkości aktualizacji.

# 2. Podstawy pomiaru wydajności — 55 min.

**Słowa kluczowe**
pomiar, metryka

**Cele nauczania – podstawy pomiaru wydajności**

**2.1. Typowe metryki zbierane podczas testowania wydajnościowego**
PTFL-2.1.1. (K2) Kandydat rozumie typowe metryki zbierane podczas testów wydajnościowych.

**2.2. Agregowanie wyników z testów wydajnościowych**
PTFL-2.1.2. (K2) Kandydat potrafi wyjaśnić cel agregowania wyników testów wydajnościowych.

**2.3. Kluczowe źródła metryk wydajności**
PTFL-2.1.3. (K2) Kandydat rozumie kluczowe źródła metryk wydajności.

**2.4. Typowe wyniki testu wydajnościowego**
PTFL-2.1.4. (K1) Kandydat potrafi przywołać typowe wyniki testu wydajnościowego.

## 2.1. Typowe metryki zbierane podczas testowania wydajnościowego

### 2.1.1. Dlaczego metryki wydajności są potrzebne

Dokładne pomiary i metryki, które pochodzą z tych pomiarów, są niezbędne do określenia celów testowania wydajnościowego i do oceny wyników testów wydajnościowych. Nie należy przeprowadzać testów wydajnościowych bez uprzedniego zrozumienia, które pomiary i metryki są potrzebne. W przypadku zignorowania tej porady występują następujące ryzyka projektowe:

- nie wiadomo, czy poziomy wydajności są akceptowalne, aby osiągnąć cele operacyjne,
- wymagania dotyczące wydajności nie są zdefiniowane w sposób mierzalny,
- może nie być możliwe zidentyfikowanie trendów, które mogą przewidywać niższe poziomy wydajności,
- rzeczywistych wyników testu wydajnościowego nie można ocenić poprzez porównanie ich z podstawowym zestawem miar wydajności, które definiują akceptowalną i/lub nieakceptowalną wydajność,
- wyniki testów wydajnościowych są oceniane na podstawie subiektywnej opinii jednej lub więcej osób,
- wyniki dostarczone przez narzędzie do testów wydajnościowych są niezrozumiałe.

### 2.1.2. Gromadzenie pomiarów i wskaźników wydajności

Jak w przypadku każdej formy pomiaru, możliwe jest precyzyjne uzyskiwanie i wyrażanie metryk. W związku z tym wszelkie metryki i pomiary opisane w tej sekcji można i należy zdefiniować tak, aby miały znaczenie w określonym kontekście. Jest to kwestia wykonania wstępnych testów i nauczenia się, które metryki wymagają dalszego dopracowania, a które należy dodać.

Na przykład metryka czasu odpowiedzi prawdopodobnie znajdzie się w dowolnym zestawie metryk wydajności. Jednak aby metryka czasu odpowiedzi dawała sensowne i dające się wykorzystać wyniki, będzie musiała zostać precyzyjnie zdefiniowana pod względem pory dnia, liczby współbieżnych użytkowników, ilości przetwarzanych danych i tak dalej.

Metryki zebrane w określonym teście wydajnościowym będą się różnić w zależności od:

- kontekstu biznesowego (procesów biznesowych, zachowania klientów i użytkowników oraz oczekiwań interesariuszy),
- kontekstu operacyjnego (technologii i sposobu jej wykorzystania),
- celów testów.

Na przykład metryki wybrane do testowania wydajnościowego międzynarodowej witryny e-commerce będą się różnić od tych wybranych do testowania wydajnościowego systemu wbudowanego używanego do kontrolowania funkcjonalności urządzeń medycznych.

Powszechnym sposobem kategoryzacji pomiarów i metryk wydajności jest uwzględnienie środowiska technicznego, środowiska biznesowego lub środowiska operacyjnego, w którym wymagana jest ocena wydajności.

Kategorie pomiarów i metryk podane poniżej to te, które są powszechnie uzyskiwane podczas testów wydajnościowych.

**Środowisko techniczne**

Metryki wydajności będą się różnić w zależności od rodzaju środowiska technicznego, jak pokazano na poniższej liście:

- webowe,
- mobilne,
- Internet rzeczy (IoT),
- urządzenia desktopowe,
- przetwarzanie po stronie serwera,
- mainframe,
- bazy danych,
- sieci,
- charakter oprogramowania działającego w środowisku (np. wbudowane).

Metryki zawierają:

- czas odpowiedzi (np. *per* transakcję, *per* współbieżni użytkownicy, czasy wczytywania strony),

- wykorzystanie zasobów (np. CPU, pamięć, przepustowość sieci, opóźnienie sieci, dostępne miejsce na dysku, szybkość operacji wejścia/wyjścia, wątki bezczynne i zajęte),
- przepustowość kluczowej transakcji (czyli liczba transakcji, które można przetworzyć w danym okresie),
- czas przetwarzania wsadowego (np. czas oczekiwania, czas przepustowości, czas odpowiedzi bazy danych, czas zakończenia),
- liczba błędów wpływających na wydajność,
- czas zakończenia (np. tworzenie, odczytywanie, aktualizowanie i usuwanie danych), obciążenie w tle współdzielonych zasobów (szczególnie w środowiskach zwirtualizowanych),
- metryki oprogramowania (np. złożoność kodu).

**Środowisko biznesowe**

Z biznesowego lub funkcjonalnego punktu widzenia metryki wydajności mogą obejmować:

- efektywność procesów biznesowych (np. szybkość wykonywania całego procesu biznesowego, w tym normalnych, alternatywnych i wyjątkowych przepływów przypadków użycia),
- przepustowość danych, transakcji i innych wykonanych jednostek pracy (np. zamówienia przetwarzane na godzinę, wiersze danych dodawane na minutę),
- zgodność z umową o gwarantowanym poziomie świadczenia usług (SLA – *Service Level Agreement*) lub wskaźniki naruszeń (np. naruszenia SLA na jednostkę czasu),
- zakres użytkowania (np. odsetek użytkowników globalnych lub krajowych wykonujących zadania w danym czasie),
- współbieżność użytkowania (np. liczba użytkowników współbieżnie wykonujących zadanie),
- czas użytkowania (np. liczba zamówień przetworzonych w okresach szczytowego obciążenia).

**Środowisko operacyjne**

Operacyjny aspekt testowania wydajnościowego koncentruje się na zadaniach, które z natury nie są uważane za przeznaczone dla użytkownika. Należą do nich:

- procesy operacyjne (np. czas potrzebny na uruchomienie środowiska, kopie zapasowe, czasy wyłączenia i wznowienia),
- przywracanie systemu (np. czas potrzebny na przywrócenie danych z kopii zapasowej),
- alerty i ostrzeżenia (np. czas potrzebny systemowi na wysłanie alertu lub ostrzeżenia).

### 2.1.3. Wybieranie metryk wydajności

Należy zauważyć, że zbieranie większej liczby metryk niż jest to wymagane niekoniecznie jest dobrą rzeczą. Każda wybrana metryka wymaga środków do spójnego gromadzenia i raportowania. Ważne jest zdefiniowanie możliwego do uzyskania zestawu metryk, które wspierają cele testów wydajnościowych.

Na przykład podejście cel-pytanie-metryka (ang. *Goal-Question-Metric* - GQM) jest pomocnym sposobem dostosowania metryk do celów wydajności. Chodzi o to, aby najpierw ustalić cele, a następnie zadawać pytania, aby wiedzieć, kiedy te cele zostaną osiągnięte. Metryki są powiązane z każdym pytaniem, aby zapewnić wymierną odpowiedź na każde pytanie (patrz podrozdział 4.3. Sylabusa Poziomu Eksperckiego - Doskonalenie procesu testowania [ISTQB_ELTM_ITP_SYL], aby uzyskać pełniejszy opis podejścia GQM). Należy zauważyć, że podejście GQM nie zawsze jest odpowiednie do procesu testowania wydajnościowego. Na przykład niektóre metryki przedstawiają stan systemu i nie są bezpośrednio powiązane z celami.

Ważne jest, aby zdać sobie sprawę, że po zdefiniowaniu i uchwyceniu początkowych pomiarów mogą być potrzebne dalsze pomiary i zebranie metryk, aby zrozumieć rzeczywiste poziomy wydajności i określić, gdzie mogą być potrzebne działania naprawcze.

## 2.2. Agregowanie wyników z testów wydajnościowych

Celem agregowania metryk wydajności jest ich zrozumienie i wyrażenie w sposób, który dokładnie oddaje całościowy obraz wydajności systemu. Gdy metryki wydajności są przeglądane tylko na poziomie szczegółowym, wyciągnięcie właściwych wniosków może być trudne - szczególnie dla interesariuszy biznesowych. Dla wielu interesariuszy głównym zagadnieniem jest to, czy czas odpowiedzi systemu, witryny internetowej lub innego obiektu testowego mieści się w dopuszczalnych granicach.

Po dokładniejszym zrozumieniu metryk wydajności można je zebrać, aby:

- interesariusze biznesowi i projektowi mogli zobaczyć „ogólny obraz” stanu wydajności systemu,
- można było zidentyfikować trendy wydajności,
- wskaźniki wydajności mogły być podawane w sposób zrozumiały.

## 2.3. Kluczowe źródła metryk wydajności

Metoda gromadzenia metryk nie powinna mieć większego niż minimalny wpływ na wydajność systemu (znany jako „efekt próbnika”). Dodatkowo ilość, dokładność i szybkość, z jaką należy gromadzić metryki wydajności, sprawiają, że użycie narzędzia jest wymagane. Chociaż łączenie użycia wielu narzędzi nie jest rzadkością, może wprowadzić nadmiarowość w używaniu narzędzi testowych i inne problemy (patrz podrozdział 4.4.).
Istnieją trzy kluczowe źródła metryk wydajności:

**Narzędzia do testów wydajnościowych**
Wszystkie narzędzia do testowania wydajnościowego zapewniają pomiary i metryki jako wynik testu. Narzędzia mogą różnić się liczbą wyświetlanych metryk, sposobem ich wyświetlania oraz możliwością dostosowania metryk przez użytkownika do konkretnej sytuacji (patrz także podrozdział 5.1.).

Niektóre narzędzia zbierają i wyświetlają metryki wydajności w formacie tekstowym, podczas gdy bardziej zaawansowane narzędzia zbierają i wyświetlają wskaźniki wydajności graficznie w formacie pulpitu nawigacyjnego (dashboard). Wiele narzędzi oferuje możliwość eksportowania metryk w celu ułatwienia oceny testów i raportowania.

**Narzędzia do monitorowania wydajności**

Narzędzia do monitorowania wydajności są często stosowane w celu uzupełnienia możliwości raportowania narzędzi do testów wydajnościowych (patrz także podrozdział 5.1.). Ponadto narzędzia monitorujące mogą być używane do bieżącego monitorowania wydajności systemu i ostrzegania administratorów systemu o obniżonych poziomach wydajności oraz wyższych poziomach błędów i alertów systemowych. Narzędzia te mogą być również używane do wykrywania i powiadamiania w przypadku podejrzanych zachowań (takich jak ataki typu „odmowa usługi" i rozproszone ataki typu „odmowa usługi" - odpowiednio DoS i DDoS).

**Narzędzia do analizy logów**

Istnieją narzędzia, które skanują logi serwera i kompilują z nich metryki. Niektóre z tych narzędzi mogą tworzyć wykresy w celu przedstawienia graficznego widoku danych.

Błędy, alerty i ostrzeżenia są zwykle zapisywane w logach serwera. Obejmują one:

- wysokie wykorzystanie zasobów, takie jak wysokie wykorzystanie procesora, wysoki poziom zajętego miejsca na dysku i niewystarczająca przepustowość,
- błędy pamięci i ostrzeżenia, takie jak wyczerpanie pamięci,
- zakleszczenia i problemy z wielowątkowością, zwłaszcza podczas wykonywania operacji na bazach danych,
- błędy bazy danych, takie jak wyjątki SQL i upłynięcie limitu czasu odpowiedzi zapytania SQL.

## 2.4. Typowe wyniki testu wydajnościowego

W testowaniu funkcjonalnym, szczególnie podczas weryfikacji określonych wymagań funkcjonalnych lub elementów funkcjonalnych historyjek użytkownika, oczekiwane wyniki można zwykle jasno zdefiniować, a wyniki zinterpretować w celu ustalenia, czy test przeszedł pomyślnie, czy też nie. Na przykład miesięczny raport sprzedaży pokazuje poprawną lub niepoprawną sumę.

Podczas gdy testy sprawdzające funkcjonalność często korzystają z dobrze zdefiniowanych wyroczni testowych, testom wydajnościowym często brakuje tego źródła informacji. Nie tylko interesariusze są nieustannie słabi w formułowaniu wymagań dotyczących wydajności, ale wielu analityków biznesowych i właścicieli produktów nie radzi sobie z ich pozyskiwaniem. Testerzy często otrzymują ograniczone wytyczne dotyczące określenia oczekiwanych wyników testów.

Podczas oceny wyników testów wydajnościowych ważne jest, aby dokładnie przyjrzeć się wynikom. Wstępne surowe wyniki mogą być mylące, ponieważ błędy wydajności są ukryte pod pozornie dobrymi wynikami ogólnymi. Na przykład wykorzystanie zasobów może być znacznie poniżej 75% dla wszystkich kluczowych potencjalnych zasobów stanowiących wąskie gardło, ale przepustowość lub czas odpowiedzi kluczowych transakcji lub przypadków użycia są o rząd wielkości zbyt wolne.

Konkretne wyniki do oceny różnią się w zależności od przeprowadzanych testów i często obejmują te omówione w podrozdziale 2.1.

# 3. Testowanie wydajnościowe w cyklu życia oprogramowania — 195 min.

**Słowa kluczowe**
metryka, ryzyko, cykl życia oprogramowania, log testów

**Cele nauczania – testowanie wydajnościowe w cyklu życia oprogramowania**

**3.1. Podstawowe czynności w zakresie testowania wydajnościowego**
PTFL-3.1.1. (K2) Kandydat rozumie znaczenie podstawowych czynności w zakresie testowania wydajnościowego.

**3.2. Ryzyka wydajności dla różnych architektur**
PTFL-3.1.2. (K2) Kandydat potrafi wyjaśnić typowe kategorie ryzyk wydajności dla różnych architektur.

**3.3. Ryzyka wydajności w całym cyklu życia oprogramowania**
PTFL-3.1.3. (K4) Kandydat potrafi przeanalizować ryzyka wydajności dla danego produktu w całym cyklu życia oprogramowania.

**3.4. Czynności w zakresie testowania wydajnościowego**
PTFL-3.1.4. (K4) Kandydat potrafi przeanalizować dany projekt, aby określić odpowiednie czynności testowania wydajnościowego dla każdego etapu cyklu życia oprogramowania.

## 3.1. Podstawowe czynności w zakresie testowania wydajnościowego

Testowanie wydajnościowe ma charakter iteracyjny. Każdy test zapewnia cenny wgląd w wydajność aplikacji i systemu. Informacje zebrane z jednego testu służą do poprawiania lub optymalizacji aplikacji i parametrów systemu. Następna iteracja testu pokaże wtedy wyniki modyfikacji i tak dalej, aż do osiągnięcia celów testu.

Testy wydajnościowe są zgodne z procesem testowym ISTQB® [ISTQB_FL_SYL].

**Planowanie testów**

Planowanie testów jest szczególnie ważne w przypadku testów wydajnościowych ze względu na konieczność przydzielenia środowisk testowych, danych testowych, narzędzi i zasobów ludzkich. Ponadto jest to czynność, w której ustalany jest zakres testów wydajnościowych.

Podczas planowania testów kończy się czynności związane z identyfikacją ryzyka i analizą ryzyka, a istotne informacje są aktualizowane w dokumentacji planowania testów (np. plan testów, plan testów jednego poziomu). Podobnie jak planowanie testów jest weryfikowane

i modyfikowane w razie potrzeby, tak samo ryzyko, poziomy ryzyka i stan ryzyka są modyfikowane w celu odzwierciedlenia zmian warunków ryzyka.

**Monitorowanie testów i nadzór nad testami**

W celu zaplanowania podjęcia czynności w przypadku napotkania problemów, które mogą wpłynąć na efektywność wydajności, definiowane są pomiary kontrolne. Dotyczy to problemów takich jak:

- zwiększenie zdolności do generowania obciążenia, jeżeli infrastruktura nie generuje żądanych obciążeń, jak zaplanowano dla poszczególnych testów wydajnościowych,
- zmieniony, nowy lub wymieniony sprzęt,
- zmiany w komponentach sieci,
- zmiany w implementacji oprogramowania.

Aby sprawdzić spełnienie kryteriów zakończenia oceniane są cele testu wydajnościowego.

**Analiza testów**

Efektywne testy wydajnościowe opierają się na analizie wymagań wydajnościowych, celów testów, umów o gwarantowanym poziomie świadczenia usług (ang. SLA – *Service Level Agreement*), architektury IT, modeli procesów i innych elementów, które składają się na podstawę testów. To działanie może być wspierane przez modelowanie i analizę wymagań i/lub zachowania zasobów systemowych przy użyciu arkuszy kalkulacyjnych lub narzędzi do planowania wydajności.

Identyfikowane są określone warunki testowe, takie jak: poziomy obciążenia, warunki czasowe i transakcje do przetestowania. Następnie ustala się wymagany(-e) typ(y) testu(-ów) wydajnościowych (np. testy obciążeniowe, przeciążeniowe, skalowalności).

**Projektowanie testów**

Projektowane są przypadki testowe dla testów wydajnościowych. Są one generalnie tworzone w formie modułowej, dzięki czemu mogą być używane jako elementy składowe większych, bardziej złożonych testów wydajnościowych (patrz podrozdział 4.2.).

**Implementacja testów**

W fazie implementacji przypadki testowe dla testów wydajnościowych są uporządkowane w procedurach testów wydajnościowych. Te procedury testów wydajnościowych powinny odzwierciedlać kroki zwykle podejmowane przez użytkownika i inne czynności funkcjonalne, które mają być uwzględnione podczas testów wydajnościowych.

Czynnością związaną z implementacją testów jest ustanowienie i/lub resetowanie środowiska testowego przed każdym wykonaniem testu. Ponieważ testowanie wydajnościowe jest zwykle oparte na danych, potrzebny jest proces w celu ustalenia danych testowych, które są

reprezentatywne dla rzeczywistych danych produkcyjnych pod względem wielkości i rodzaju, aby można było zasymulować wykorzystanie produkcyjne.

**Wykonywanie testów**
Wykonywanie testów ma miejsce podczas przeprowadzania testów wydajnościowych, często przy użyciu narzędzi do testów wydajnościowych. Wyniki testów są oceniane w celu określenia, czy wydajność systemu spełnia wymagania i inne określone cele. Wszelkie defekty są zgłaszane.

**Ukończenie testów**
Wyniki testów wydajnościowych są przedstawiane interesariuszom (np. architektom, menedżerom, właścicielom produktów) w sumarycznym raporcie z testów. Wyniki są wyrażane za pomocą metryk, które często są agregowane w taki sposób, aby uprościć zrozumienie wyników testów. Do zaprezentowania wyników testów wydajnościowych często wykorzystywane są wizualne sposoby raportowania, takie jak pulpity nawigacyjne, które są łatwiejsze do zrozumienia niż metryki tekstowe (np. dane liczbowe).

Testy wydajnościowe jest często uważane za czynność ciągłą, ponieważ są wykonywane wielokrotnie i na wszystkich poziomach testów (testowanie komponentów, integracji, systemu, integracji systemów i testów akceptacyjnych). Pod koniec określonego cyklu testów wydajnościowych może być osiągnięty punkt zakończenia testów, w którym zaprojektowane testy, artefakty narzędzi testowych (przypadki testowe i procedury testowe), dane testowe i inne testalia są archiwizowane lub przekazywane innym testerom do późniejszego wykorzystania podczas czynności konserwacyjnych systemu.

## 3.2. Kategorie ryzyk wydajności dla różnych architektur

Jak wspomniano powyżej, wydajność aplikacji lub systemu znacznie się różni w zależności od architektury, aplikacji i środowiska hosta. Chociaż nie jest możliwe przedstawienie pełnej listy ryzyk dotyczących wydajności dla wszystkich systemów, poniższa lista zawiera kilka typowych rodzajów ryzyk związanych z określonymi architekturami.

**Systemy uruchamiane na pojedynczym komputerze**

Są to systemy lub aplikacje działające w całości na jednym niezwirtualizowanym komputerze. Wydajność może się pogorszyć z powodu:

- nadmiernego zużycia zasobów, w tym wycieków pamięci, działania innych procesów w tle takich jak oprogramowanie zabezpieczające, powolnych podsystemów pamięci masowej (np. wolne urządzenia zewnętrzne lub fragmentacja dysku) oraz złego zarządzania systemem operacyjnym,
- nieefektywnej implementacji algorytmów, które nie wykorzystują dostępnych zasobów (np. głównej pamięci) i w rezultacie działają wolniej niż jest to wymagane.

**Systemy wielowarstwowe**

Są to systemy systemów działających na wielu serwerach, z których każdy wykonuje określony zestaw zadań takich jak serwer bazy danych, serwer aplikacji i serwer prezentacji. Każdy serwer jest oczywiście komputerem i podlega podanym wcześniej ryzykom. Ponadto wydajność może spaść z powodu słabego lub nieskalowalnego projektu bazy danych, wąskich gardeł w sieci oraz niewystarczającej przepustowości lub pojemności na jednym serwerze.

**Systemy rozproszone**

Są to systemy systemów podobne do architektury wielowarstwowej, ale różne serwery mogą zmieniać się dynamicznie, na przykład system e-commerce, który uzyskuje dostęp do różnych zasobów baz danych w zależności od położenia geograficznego osoby składającej zamówienie. Oprócz zagrożeń związanych z architekturami wielowarstwowymi, architektura ta może mieć problemy z wydajnością z powodu krytycznych przepływów pracy lub przepływów danych do, z lub przez zawodne lub nieprzewidywalne serwery zdalne, zwłaszcza gdy takie serwery mają okresowe problemy z połączeniami lub sporadyczne okresy intensywnego obciążenia.

**Systemy zwirtualizowane**

Są to systemy, w których sprzęt fizyczny obsługuje wiele komputerów wirtualnych. Te maszyny wirtualne mogą obsługiwać systemy i aplikacje uruchamiane na jednym komputerze, a także serwery, które są częścią wielowarstwowej lub rozproszonej architektury. Ryzyka dla wydajności, które wynikają w szczególności z wirtualizacji, obejmują nadmierne obciążenie sprzętu na wszystkich maszynach wirtualnych lub niewłaściwą konfigurację maszyny hostującej maszynę wirtualną, co skutkuje niewystarczającymi zasobami.

**Systemy dynamiczne / oparte na chmurze**

Są to systemy, które oferują możliwość skalowania na żądanie, zwiększając wydajność wraz ze wzrostem poziomu obciążenia. Systemy te są zwykle rozproszonymi i zwirtualizowanymi systemami wielowarstwowymi, aczkolwiek z samoskalowującymi się funkcjami, zaprojektowanymi specjalnie w celu złagodzenia niektórych ryzyk wydajności związanych z tymi architekturami. Istnieje jednak ryzyko związane z błędami w prawidłowej konfiguracji tych funkcji podczas początkowej konfiguracji lub jej kolejnych aktualizacji.

**Systemy klient-serwer**

Są to systemy działające na komputerze typu klient, które komunikują się za pośrednictwem interfejsu użytkownika z pojedynczym serwerem, serwerem wielowarstwowym lub serwerem rozproszonym. W związku z tym, że na kliencie działa kod, ryzyko związane z pojedynczym komputerem odnosi się do tego kodu, podczas gdy problemy związane z serwerem opisane powyżej wciąż pozostają aktualne. Ponadto istnieje ryzyko związane z wydajnością z powodu problemów z szybkością i niezawodnością połączenia, przeciążeniem sieci w punkcie połączenia klienta (np. publiczna sieć Wi-Fi) oraz potencjalnymi problemami związanymi

z zaporami firewall, kontrolą pakietów i systemem równoważeniem obciążenia (ang. *load balancing*) serwera.

**Aplikacje mobilne**

Są to aplikacje działające na smartfonie, tablecie lub innym urządzeniu mobilnym. Takie aplikacje podlegają ryzykom wymienionym w przypadku aplikacji typu klient-serwer i aplikacji przeglądarkowych (webowych). Ponadto problemy z wydajnością mogą wynikać z ograniczonych i zmieniających się zasobów oraz połączenia z siecią dostępnego na urządzeniu mobilnym (na które może mieć wpływ lokalizacja, żywotność baterii, stan naładowania, dostępna pamięć na urządzeniu i temperatura). W przypadku aplikacji korzystających z czujników urządzenia lub radia, takich jak akcelerometry lub Bluetooth, problemy mogą być spowodowane przez powolny przepływ danych z tych źródeł. Wreszcie, aplikacje mobilne często wchodzą w intensywne interakcje z innymi lokalnymi aplikacjami mobilnymi i zdalnymi usługami internetowymi, z których każda może potencjalnie stać się wąskim gardłem wydajności.

**Wbudowane systemy czasu rzeczywistego**

Są to systemy, które działają z przedmiotami codziennego użytku lub nawet je kontrolują, takie jak samochody (np. systemy rozrywki i inteligentne układy hamulcowe), windy, sygnalizacja świetlna, systemy Ogrzewania, Wentylacji i Klimatyzacji (ang. HVAC - *Heating, Ventilation and Air Conditioning*) i nie tylko. W tych systemach często występują ryzyka związane z urządzeniami mobilnymi, w tym (coraz częściej) problemy dotyczące łączności, ponieważ urządzenia te są połączone z Internetem. Jednak zmniejszona wydajność mobilnej gry wideo zwykle nie stanowi zagrożenia dla użytkownika, podczas gdy takie spowolnienia w układzie hamulcowym pojazdu mogą okazać się katastrofalne.

**Aplikacje mainframe**

Są to aplikacje - w wielu przypadkach aplikacje sprzed dziesięcioleci - obsługujące często krytyczne funkcje biznesowe w centrach danych, czasami poprzez przetwarzanie wsadowe. Większość z nich jest dość przewidywalna i szybka, gdy jest używana zgodnie z pierwotnym projektem, ale wiele z nich jest teraz dostępnych za pośrednictwem interfejsów API, usług internetowych lub baz danych, co może skutkować nieoczekiwanymi obciążeniami, które wpływają na przepustowość istniejących aplikacji.

Należy pamiętać, że każda konkretna aplikacja lub system może zawierać dwie lub więcej architektur wymienionych powyżej, co oznacza, że wszystkie istotne ryzyka będą dotyczyć tej aplikacji lub systemu. W rzeczywistości, biorąc pod uwagę Internet rzeczy (IoT) i przyrost aplikacji mobilnych - dwa obszary, w których regułą są ekstremalne poziomy interakcji i połączenia - możliwe jest, że wszystkie architektury są obecne w jakiejś formie w aplikacji, a zatem mogą wystąpić wszystkie ryzyka.

Chociaż architektura jest z pewnością ważną decyzją techniczną mającą głęboki wpływ na ryzyka wydajnościowe, inne decyzje techniczne również wpływają na ryzyka i je tworzą.

Na przykład wycieki pamięci są bardziej powszechne w językach, które umożliwiają bezpośrednie zarządzanie stosem pamięci, takich jak C i C++, a problemy z wydajnością są inne w przypadku relacyjnych i nierelacyjnych baz danych. Takie decyzje przedłużają się aż do zaprojektowania poszczególnych funkcji lub metod (np. wybór algorytmu rekurencyjnego w przeciwieństwie do iteracyjnego). Odnosząc to do testerów, możliwość poznania lub nawet wpływania na takie decyzje będzie się różnić w zależności od ról i obowiązków testerów w organizacji i cyklu życia oprogramowania.

## 3.3. Ryzyka wydajności w całym cyklu życia oprogramowania

Proces analizy ryzyk w odniesieniu do jakości produktu oprogramowania został omówiony w sposób ogólny w różnych sylabusach ISTQB® (np. patrz [ISTQB_FL_SYL] i [ISTQB_ALTM_SYL]). Można również znaleźć dyskusje na temat konkretnych ryzyk i rozważania związane z określonymi cechami jakości (np. [ISTQB_UT_SYL]) oraz z perspektywą biznesową lub techniczną (np. patrz odpowiednio [ISTQB_ALTA_SYL] i [ISTQB_ALTTA_SYL]). W tej części skupiono się na ryzyku związanym z wydajnością w odniesieniu do jakości produktu, w tym na zmianach dotyczących procesu, uczestników i uwarunkowań.

W przypadku ryzyk związanych z jakością wydajności danego produktu, proces jest zdefiniowany w następujący sposób:

1. Zidentyfikowanie ryzyka dla jakości produktu i skoncentrowanie się na takich cechach jak: zachowanie w czasie, wykorzystanie zasobów i przepustowość.
2. Ocena zidentyfikowanego ryzyka oraz upewnienie się, że uwzględniono odpowiednie kategorie architektury (patrz podrozdział 3.2.). Ocena ogólnego poziomu ryzyka dla każdego zidentyfikowanego ryzyka pod względem prawdopodobieństwa i wpływu, przy zastosowaniu jasno zdefiniowanych kryteriów.
3. Podjęcie odpowiedniego działania ograniczającego ryzyko dla każdego zidentyfikowanego ryzyka w oparciu o jego charakter i poziom.
4. Bieżące zarządzanie ryzykiem, aby mieć pewność, że ryzyko zostało odpowiednio ograniczone przed wydaniem oprogramowania.

Podobnie jak w przypadku ogólnej analizy ryzyka jakościowego, uczestnikami tego procesu powinni być zarówno interesariusze biznesowi, jak i techniczni. W przypadku analizy ryzyka związanego z wydajnością w gronie interesariuszy biznesowych muszą znaleźć się ci, którzy mają szczególną świadomość tego, jak problemy z wydajnością w środowisku produkcyjnym wpłyną w rzeczywistości na klientów, użytkowników, firmę i inne zainteresowane strony. Interesariusze biznesowi muszą zdawać sobie sprawę z tego, że zakładane użytkowanie, krytyczność biznesowa, społeczna lub bezpieczeństwo, potencjalne szkody finansowe i/lub reputacyjne, odpowiedzialność cywilna lub karna oraz podobne czynniki wpływają na ryzyko z perspektywy biznesowej, stwarzając ryzyko i wpływając na skutki awarii.

Ponadto wśród interesariuszy technicznych muszą znajdować się ci, którzy mają dogłębną wiedzę na temat wpływu odpowiednich wymagań, architektury, projektu i decyzji wdrożeniowych. Interesariusze techniczni muszą zdawać sobie sprawę z tego, że decyzje dotyczące architektury, projektu i wdrażania wpływają na ryzyko związane z wydajnością z technicznego punktu widzenia, tworząc ryzyko i wpływając na prawdopodobieństwo wystąpienia defektów.

Wybrany konkretny proces analizy ryzyka powinien mieć odpowiedni poziom sformalizowania i rygoru. W przypadku ryzyk związanych z wynikami szczególnie ważne jest, aby proces analizy ryzyka rozpoczął się wcześnie i był regularnie powtarzany. Innymi słowy, tester powinien unikać całkowitego polegania na testach wydajnościowych przeprowadzanych pod koniec testów systemowych i testów integracji systemów. Wiele projektów, zwłaszcza tych większych i bardziej złożonych typu “system zbudowany z systemów”, spotkało się z niefortunnymi niespodziankami ze względu na późne wykrycie defektów w zakresie wydajności, które wynikały z wymagań, projektu, architektury i decyzji wdrożeniowych podjętych na wczesnym etapie projektu. Dlatego należy położyć nacisk na iteracyjne podejście do identyfikacji, oceny, łagodzenia i zarządzania ryzykiem związanym z wydajnością w całym cyklu życia oprogramowania.

Na przykład, jeśli duże ilości danych będą obsługiwane za pośrednictwem relacyjnej bazy danych, niska wydajność łączenia wiele do wielu z powodu błędnego projektu bazy danych może ujawnić się tylko podczas testów dynamicznych z dużymi, testowymi zestawami danych, takimi jak te używane podczas testów systemowych. Jednakże dokładny przegląd techniczny, w którym uczestniczą doświadczeni inżynierowie baz danych, może skutkować zdiagnozowaniem problemów przed wdrożeniem bazy danych. Po takim przeglądzie, w podejściu iteracyjnym, ryzyka są identyfikowane i ponownie szacowane.

Ponadto ograniczanie ryzyka i zarządzanie nim musi obejmować i mieć wpływ na cały proces tworzenia oprogramowania, a nie tylko na testowanie dynamiczne. Na przykład, gdy krytyczne decyzje związane z wydajnością, takie jak oczekiwana liczba transakcji lub współbieżnych użytkowników, nie mogą zostać określone na wczesnym etapie projektu, ważne jest, aby decyzje projektowe i architektoniczne umożliwiały wysoce zmienną skalowalność (np. chmurowe zasoby obliczeniowe dostępne na żądanie). Umożliwia to podejmowanie decyzji dotyczących łagodzenia ryzyka na wczesnym etapie.

Prawidłowo wdrożony proces inżynierii wydajności może pomóc zespołom projektowym uniknąć późnego wykrycia krytycznych defektów wydajności podczas testowania na wyższych poziomach testów, takich jak testy integracji systemów lub testy akceptacyjne użytkownika. Defekty wydajności wykryte na późnym etapie projektu mogą być niezwykle kosztowne, a nawet mogą prowadzić do anulowania całych projektów.

Podobnie jak w przypadku każdego rodzaju ryzyka jakościowego, nigdy nie można całkowicie uniknąć ryzyka związanego z wydajnością, tj. zawsze będzie istnieć pewne ryzyko awarii środowiska produkcyjnego związanej z wydajnością. Dlatego proces zarządzania ryzykiem musi obejmować zapewnienie realistycznej i szczegółowej oceny pozostałego poziomu ryzyka dla biznesu i interesariuszy technicznych zaangażowanych w proces. Na przykład zwykłe

powiedzenie: „Tak, klienci nadal mogą doświadczyć dużych opóźnień podczas realizacji transakcji” nie jest pomocne, ponieważ nie daje pojęcia o tym, jaki stopień ograniczenia ryzyka wystąpił, ani też o pozostałym ryzyku. Zamiast tego, pomocnym w zrozumieniu faktycznego statusu ryzyka może być dostarczenie jasnego wglądu w odsetek klientów, którzy mogą doświadczyć opóźnień równych lub przekraczających określone progi.

## 3.4. Czynności w zakresie testowania wydajnościowego

Czynności związane z testowaniem wydajnościowym będą organizowane i wykonywane w różny sposób, w zależności od aktualnej fazy cyklu życia oprogramowania.

**Sekwencyjne modele wytwarzania oprogramowania**

Idealną praktyką testowania wydajnościowego w sekwencyjnych modelach wytwarzania oprogramowania jest włączenie kryteriów wydajności jako części kryteriów akceptacji, które są definiowane na początku projektu. Wspierając punkt widzenia cyklu życia na czynności procesu testowania wydajnościowego, powinny być one przeprowadzane w ciągu całego cyklu życia oprogramowania. W miarę postępów projektu każda kolejna czynność dotycząca testów wydajnościowych powinna opierać się na elementach zdefiniowanych we wcześniejszych działaniach, jak opisano poniżej.

- Koncepcja - sprawdzenie, czy cele wydajności systemu są zdefiniowane jako kryteria akceptacji dla projektu.
- Wymagania - sprawdzenie, czy wymagania dotyczące wydajności są zdefiniowane i prawidłowo reprezentują potrzeby interesariuszy.
- Analiza i projekt - sprawdzenie, czy projekt systemu odzwierciedla wymagania dotyczące wydajności.
- Kodowanie/Implementacja - sprawdzenie, czy kod jest wydajny i odzwierciedla wymagania oraz projekt pod względem wydajności.
- Testowanie komponentów - przeprowadzanie testów wydajnościowych na poziomie komponentów.
- Testowanie integracji komponentów - przeprowadzanie testów wydajnościowych na poziomie integracji komponentów.
- Testy systemowe - przeprowadzanie testów wydajnościowych na poziomie systemu, które obejmują sprzęt, oprogramowanie, procedury i dane reprezentatywne dla środowiska produkcyjnego. Interfejsy systemu mogą być symulowane, pod warunkiem, że dają rzeczywistą reprezentację wydajności.
- Testy integracji systemów - przeprowadzenie testów wydajnościowych całego systemu, który jest reprezentatywny dla środowiska produkcyjnego.
- Testy akceptacyjne - sprawdzenie, czy wydajność systemu spełnia pierwotnie określone potrzeby użytkownika i kryteria akceptacji.

**Iteracyjne i przyrostowe modele wytwarzania oprogramowania**

W modelach wytwarzania oprogramowania takich jak Agile, testowanie wydajnościowe jest również postrzegane jako działanie iteracyjne i przyrostowe (patrz [ISTQB_FL_AT]). Testowanie wydajnościowe może odbywać się jako część pierwszej iteracji lub jako iteracja poświęcona całkowicie testom wydajnościowym. Jednak w przypadku tych modeli cyklu życia wykonywanie testów wydajnościowych może być przeprowadzane przez oddzielny zespół, którego zadaniem jest testowanie wydajnościowe.

W iteracyjnych i przyrostowych cyklach życia oprogramowania zwykle praktykuje się podejście ciągłej integracji (CI – *Continuous Integration*), co ułatwia wysoce zautomatyzowane wykonywanie testów. Najczęstszym celem testowania w CI jest wykonanie testów regresji i zapewnienie stabilności każdej kompilacji. Testowanie wydajnościowe może być częścią testów automatycznych wykonywanych w CI, jeśli testy są zaprojektowane w taki sposób, aby były wykonywane na poziomie kompilacji. Jednak w przeciwieństwie do funkcjonalnych testów automatycznych istnieją dodatkowe problemy, takie jak:

- Konfiguracja środowiska dla testów wydajnościowych - często wymaga to środowiska testowego, które jest dostępne na żądanie, takiego jak środowisko testowe wydajności oparte na chmurze.
- Określenie, które testy wydajnościowe zautomatyzować w CI - ze względu na wąskie ramy czasowe dostępne dla testów CI, testy wydajnościowe CI mogą być podzbiorem bardziej rozbudowanych testów wydajnościowych, które są przeprowadzane przez zespół specjalistów w innym czasie podczas iteracji.
- Tworzenie testów wydajnościowych dla CI - głównym celem testów wydajnościowych w ramach CI jest zapewnienie, że zmiana nie wpłynie negatywnie na wydajność. W zależności od zmian wprowadzonych dla danej kompilacji mogą być wymagane nowe testy wydajnościowe.
- Wykonywanie testów wydajnościowych na częściach aplikacji lub systemu - często wymaga to od narzędzi i środowisk testowych możliwości szybkiego testowania wydajności, w tym możliwości wyboru podzbiorów odpowiednich testów.

Testowanie wydajnościowe w iteracyjnych i przyrostowych cyklach życia oprogramowania może również mieć własne działania w czynnościach cyklu życia:

- Planowanie wydania - w tym działaniu testowanie wydajności rozpatruje się z perspektywy wszystkich iteracji w wydaniu, począwszy od pierwszej, aż do ostatniej. Ryzyka związane z wydajnością są identyfikowane i oceniane oraz planowane są środki łagodzące. Często obejmuje to planowanie wszelkich ostatecznych testów wydajnościowych przed wydaniem aplikacji.
- Planowanie iteracji - w kontekście każdej iteracji testy wydajnościowe mogą być wykonywane w ramach iteracji oraz po zakończeniu każdej iteracji. Ryzyka wydajności są oceniane bardziej szczegółowo dla każdej historyjki użytkownika.
- Tworzenie historyjek użytkownika - historyjki użytkownika często stanowią podstawę wymagań w zakresie wydajności w metodykach zwinnych, z określonymi kryteriami wydajności opisanymi w powiązanych kryteriach akceptacji. Są one określane jako „niefunkcjonalne” historyjki użytkownika.

- Projektowanie testów wydajnościowych - do projektowania testów wykorzystuje się wymagania i kryteria wydajności, które są opisane w poszczególnych historyjkach użytkownika (patrz podrozdział 4.2.).
- Kodowanie/implementacja - podczas programowania testy wydajnościowe mogą być wykonywane na poziomie komponentów. Przykładem może być dostrojenie algorytmów w celu uzyskania optymalnej wydajności.
- Testowanie/ocena - podczas gdy testowanie jest zwykle wykonywane w bliskim sąsiedztwie działań programistycznych, testowanie wydajnościowe może być wykonywane jako oddzielna czynność, w zależności od zakresu i celów testów wydajnościowych podczas iteracji. Na przykład, jeśli celem testów wydajnościowych jest przetestowanie wydajności iteracji jako kompletnego zestawu historyjek użytkownika, potrzebny będzie szerszy zakres testów wydajnościowych niż w przypadku testowania wydajnościowego pojedynczej historyjki użytkownika. Można to zaplanować w dedykowanej iteracji do testów wydajnościowych.
- Dostawa - ponieważ dostarczenie wprowadzi aplikację do środowiska produkcyjnego, wydajność będzie musiała być monitorowana w celu określenia, czy aplikacja osiąga pożądany poziom wydajności w rzeczywistym użyciu.

**Oprogramowanie do powszechnej sprzedaży (COTS od ang. *Commercial off-the-Shelf*) i inne modele dostawców / nabywców**

Wiele organizacji nie wytwarza samodzielnie aplikacji i systemów, ale zamiast tego jest w stanie nabywać oprogramowanie od dostawców lub z projektów *open source*. W takich modelach dostawcy / nabywcy wydajność jest ważnym, branym pod uwagę, aspektem, który wymaga testowania zarówno z perspektywy dostawcy (sprzedawcy / dewelopera), jak i nabywcy (klienta).

Niezależnie od źródła aplikacji, często obowiązkiem klienta jest sprawdzenie, czy wydajność spełnia jego wymagania. W przypadku niestandardowego oprogramowania opracowanego przez dostawcę, wymagania dotyczące wydajności i związane z nimi kryteria akceptacji powinny być określone w ramach umowy między dostawcą a klientem. W przypadku aplikacji COTS, klient ponosi wyłączną odpowiedzialność za przetestowanie działania produktu w realistycznym środowisku testowym, przed wdrożeniem.

# 4. Zadania testowania wydajnościowego – 475 min.

## Słowa kluczowe

współbieżność, profil obciążenia, generowanie obciążenia, profil operacyjny, spowolnienie, przyspieszenie, system systemów, przepustowość systemu, plan testów, czas do namysłu, użytkownik wirtualny

## Cele nauczania – zadania testowania wydajnościowego

**4.1. Planowanie**

PTFL-4.1.1. (K4) Kandydat potrafi wyznaczyć cele testów wydajnościowych na podstawie odpowiednich informacji.
PTFL-4.1.2. (K4) Kandydat potrafi przedstawić plan testów wydajnościowych, który uwzględnia cele wydajności dla danego projektu.
PTFL-4.1.3. (K4) Kandydat potrafi utworzyć prezentację, która umożliwi różnym interesariuszom zrozumienie uzasadnienia planowanych testów wydajnościowych.

**4.2. Analiza, projektowanie i wdrażanie**

PTFL-4.2.1. (K2) Kandydat potrafi podać przykłady typowych protokołów spotykanych podczas testowania wydajnościowego.
PTFL-4.2.2. (K2) Kandydat rozumie pojęcie transakcji w testowaniu wydajnościowym.
PTFL-4.2.3. (K4) Kandydat potrafi przeanalizować profil operacyjny pod kątem wykorzystania systemu.
PTFL-4.2.4. (K4) Kandydat potrafi utworzyć profile obciążenia pochodzące z profili operacyjnych dla określonych celów wydajności.
PTFL-4.2.5. (K4) Kandydat potrafi przeanalizować przepustowość i współbieżność podczas opracowywania testów wydajnościowych.
PTFL-4.2.6. (K2) Kandydat rozumie podstawową strukturę skryptu testu wydajnościowego.
PTFL-4.2.7. (K3) Kandydat potrafi zaimplementować skrypty testów wydajnościowych zgodne z planem i profilami obciążenia.
PTFL-4.2.8. (K2) Kandydat rozumie czynności związane z przygotowaniem testów wydajnościowych.

**4.3. Wykonanie testu**

PTFL-4.3.1. (K2) Kandydat rozumie podstawowe czynności związane z uruchamianiem skryptów testów wydajnościowych.

**4.4. Analiza wyników i raportowanie**

PTFL-4.4.1. (K4) Kandydat potrafi przeanalizować i zaraportować wyniki testów wydajnościowych oraz ich następstwa.

# 4.1. Planowanie

## 4.1.1. Wyznaczanie celów testów wydajnościowych

Wśród interesariuszy mogą znaleźć się użytkownicy i osoby z doświadczeniem biznesowym lub technicznym. Mogą mieć oni różne cele związane z testowaniem wydajnościowym. Interesariusze ustalają cele, używaną terminologię i kryteria określające, czy cel został osiągnięty.

Cele testów wydajnościowych są powiązane z różnymi typami interesariuszy. Dobrą praktyką jest rozróżnienie między celami opartymi na użytkowniku (wyznaczonymi przez użytkowników), a celami technicznymi. Cele oparte na użytkowniku(wyznaczone przez użytkownika) koncentrują się przede wszystkim na zapewnieniu satysfakcji użytkownikowi końcowemu i realizacji celów biznesowych. Ogólnie rzecz biorąc, użytkownicy są mniej

zainteresowani typami funkcji lub sposobem dostarczenia produktu. Chcą po prostu być w stanie robić to, co muszą.

Z drugiej strony cele techniczne koncentrują się na aspektach operacyjnych i udzielaniu odpowiedzi na pytania dotyczące zdolności systemu do skalowania lub tego, w jakich warunkach spadek wydajności może stać się widoczny.

Kluczowe cele testowania wydajnościowego obejmują identyfikację potencjalnych ryzyk, znajdowanie możliwości udoskonaleń oraz identyfikację niezbędnych zmian.

Podczas zbierania informacji od różnych interesariuszy należy udzielić odpowiedzi na następujące pytania:

- Jakie transakcje zostaną wykonane w teście wydajnościowym i jaki średni czas odpowiedzi jest oczekiwany?
- Jakie metryki systemu mają być rejestrowane (np. użycie pamięci, przepustowość sieci) i jakie wartości są oczekiwane?
- Jakich udoskonaleń w obszarze wydajności oczekuje w ramach tych testów w porównaniu z poprzednimi cyklami testowymi?

### 4.1.2. Plan testów wydajnościowych

Plan testów wydajnościowych (PTW) to dokument utworzony przed przeprowadzeniem jakichkolwiek testów wydajnościowych. Do PTW powinien odnosić się plan testów (patrz [ISTQB_FL_SYL]), który zawiera również odpowiednie informacje dotyczące harmonogramu testów. Dokument ten jest aktualizowany po rozpoczęciu testów wydajnościowych.

W PTW należy podać następujące informacje:

**Cel**

Plan opisuje cele, strategie i metody testów wydajnościowych. Umożliwia udzielenie odpowiedzi, w sposób wymierny, na ogólne pytanie o adekwatność i gotowość systemu do pracy pod obciążeniem.

**Cele testu**

Ogólne cele testów dotyczące wydajności, które ma zostać osiągnięty przez testowany system (ang. *System Under Test* - SUT), są wymienione dla każdego typu interesariusza (patrz sekcja 4.1.1.)

**Przegląd systemu**

Krótki opis SUT zapewni kontekst pomiaru parametrów testu wydajnościowego. Przegląd powinien zawierać ogólny (wysokopoziomowy) opis funkcjonalności testowanej pod obciążeniem.

**Rodzaje przeprowadzanych testów wydajnościowych**

Rodzaje testów wydajnościowych, które należy przeprowadzić, zostały wymienione wraz z opisem celu każdego typu (patrz podrozdział 1.2.).

**Kryteria akceptacji**

Testowanie wydajnościowe ma na celu określenie szybkości (czasu) odpowiedzi, przepustowości, niezawodności i/lub skalowalności systemu przy danym obciążeniu. Ogólnie rzecz biorąc, czas odpowiedzi jest kwestią użytkownika, przepustowość jest kwestią biznesową, a wykorzystanie zasobów jest kwestią systemu. Kryteria akceptacji powinny być ustalone dla wszystkich istotnych pomiarów i - jeśli ma to zastosowanie - odnosić się do następujących obszarów:

- ogólne cele testów wydajnościowych,
- umowy dotyczące gwarantowanego poziomu świadczenia usług (SLA),
- wartości bazowe - to zestaw metryk używanych do porównania bieżących i wcześniej osiągniętych wyników w obszarze wydajności. Umożliwia on zademonstrowanie określonej poprawy wydajności i/lub potwierdzenie spełnienia kryteriów akceptacji testu. Może zaistnieć potrzeba utworzenia początkowych wartości bazowych przy użyciu oczyszczonych danych z bazy danych.

**Dane testowe**

Dane testowe obejmują szeroki zakres danych, które należy zdefiniować dla testu wydajnościowego. Dane te mogą zawierać:

- dane konta użytkownika (np. konta użytkowników dostępne do jednoczesnego logowania),
- dane wejściowe użytkownika (np. dane, które użytkownik wprowadziłby do aplikacji w celu wykonania procesu biznesowego),
- bazę danych (np. wstępnie wypełnioną bazę danych, która jest uzupełniona danymi testowymi).

Proces tworzenia danych testowych powinien uwzględniać następujące aspekty:

- ekstrakcję danych z danych produkcyjnych,
- importowanie danych do testowanego systemu (SUT),
- tworzenie nowych danych,
- tworzenie kopii zapasowych, które można wykorzystać do przywrócenia danych podczas wykonywania nowych cykli testowych,
- maskowanie lub anonimizacja danych. Ta praktyka jest stosowana w przypadku danych produkcyjnych, które zawierają dane osobowe i jest obowiązkowa na mocy ogólnych przepisów o ochronie danych osobowych (RODO). Jednakże w testach wydajnościowych maskowanie danych powoduje pojawienie się dodatkowego ryzyka, ponieważ zamaskowane dane mogą nie mieć takich samych charakterystyk jak rzeczywiste dane.

**Konfiguracja systemu**

Sekcja PTW (plan testów wydajnościowych) dotycząca konfiguracji systemu zawiera następujące informacje techniczne:

- opis konkretnej architektury systemu, w tym serwerów (np. WWW, bazy danych, system równoważenia obciążenia [ang. *load balancer*]),
- definicję wielu poziomów,
- szczegółowe informacje na temat sprzętu komputerowego (np. rdzenie procesora, pamięć RAM, napędy półprzewodnikowe (SSD), dyski twarde (HDD)), w tym ich wersje

- szczegółowe informacje o oprogramowaniu (np. aplikacje, systemy operacyjne, bazy danych, usługi wykorzystywane do wspierania przedsiębiorstwa), w tym ich wersje,
- listę systemów zewnętrznych współpracujących z testowanym systemem oraz ich konfiguracje i wersje (np. system e-commerce z integracją z NetSuite),
- identyfikator wersji/kompilacji testowanego systemu.

**Środowisko testowe**

Środowisko testowe jest często oddzielnym środowiskiem, które naśladuje produkcję, ale na mniejszą skalę. Ta część PTW powinna zawierać opis sposobu ekstrapolacji wyników z testów wydajnościowych w celu zastosowania ich do większego środowiska produkcyjnego.
W przypadku niektórych systemów środowisko produkcyjne staje się jedyną realną opcją testowania, ale w tym przypadku należy omówić konkretne ryzyko związane z tego typu testowaniem.

Narzędzia testowe czasami znajdują się poza samym środowiskiem testowym i mogą wymagać specjalnych praw dostępu w celu interakcji z komponentami systemu. Jest to kwestia dotycząca środowiska testowego i konfiguracji.

Testy wydajnościowe można również przeprowadzić z częścią systemu, która może działać bez innych komponentów. Jest to często tańsze niż testowanie całego systemu i można je przeprowadzić, gdy tylko komponent zostanie zaimplementowany.

**Narzędzia testowe**

Ta część zawiera opis narzędzi testowych (i ich wersji), które będą używane podczas tworzenia skryptów, wykonywania i monitorowania testów wydajnościowych (patrz Rozdział 5.). Ta lista zwykle obejmuje:

- narzędzia używane do symulacji transakcji użytkowników,
- narzędzia do zapewniania obciążenia z wielu punktów w architekturze systemu (punkty obecności),
- narzędzia do monitorowania wydajności systemu, w tym te opisane powyżej w sekcji dotyczącej konfiguracji systemu.

**Profile**

Profile operacyjne zapewniają powtarzalny przepływ przez aplikację (krok po kroku) dla konkretnego wykorzystania systemu. Agregacja tych profili operacyjnych daje w wyniku profil obciążenia (powszechnie określany jako scenariusz). Więcej informacji na temat profili znajduje się w sekcji 4.2.3.

**Odpowiednie metryki**

Podczas wykonywania testów wydajnościowych można zebrać dużą liczbę pomiarów i metryk (patrz Rozdział 2). Jednak. wykonanie zbyt wielu pomiarów może utrudnić analizę, a także negatywnie wpłynąć na rzeczywistą wydajność aplikacji. Z tych powodów ważne jest, aby

zidentyfikować pomiary i metryki, które są najbardziej odpowiednie dla osiągnięcia celów testu wydajnościowego. Poniższa tabela, objaśniona bardziej szczegółowo w podrozdziale 4.4., przedstawia typowy zestaw wskaźników do testowania wydajnościowego i monitorowania. Cele testów wydajnościowych powinny być zdefiniowane dla tych metryk, jeśli jest to wymagane, dla projektu:

| **Metryki wydajności** | |
|---|---|
| **Typ** | **Metryka** |
| Status użytkownika wirtualnego | # Zaliczenie (and. *passed*)<br># Niezaliczenie (ang. *failed*) |
| Czas reakcji (odpowiedzi) transakcji | Minimalny<br>Maksymalny<br>Średni<br>90% Percentyl |
| Transakcje na sekundę | # Zaliczonych / sekundę<br># Niezaliczonych / sekundę<br># Wszystkich / sekundę |
| Trafienia (np. w bazie danych lub na serwerze internetowym) | Trafienia / sekundę<br>• Minimum<br>• Maksimum<br>• Średnia<br>• Razem |
| Przepustowość | Bity / sekundę<br>• Minimum<br>• Maksimum<br>• Średnia<br>• Razem |
| Odpowiedzi HTTP na sekundę | Odpowiedzi / sekundę<br>• Minimum<br>• Maksimum<br>• Średnia<br>• Razem<br>Kody odpowiedzi HTTP |

| **Monitorowanie wydajności** | |
|---|---|
| **Typ** | **Metryka** |
| Użycie CPU | % używanego CPU |
| Zużycie pamięci | % używanej pamięci |

**Ryzyka**

Ryzyko może obejmować obszary, które nie są mierzone w ramach testów wydajnościowych, a także ograniczenia testów wydajnościowych (np. zewnętrzne interfejsy, których nie można zasymulować, niewystarczające obciążenie, brak możliwości monitorowania serwerów). Ograniczenia środowiska testowego mogą również powodować zagrożenia (np. niewystarczające dane, zmniejszone środowisko). Więcej opisów rodzajów ryzyka można znaleźć w podrozdziałach 3.2. i 3.3.

### 4.1.3. Informowanie o testach wydajnościowych

Tester musi być w stanie przekazać wszystkim interesariuszom uzasadnienie podejścia do testów wydajnościowych i czynności, które mają być podjęte (jak wyszczególniono w Planie Testów Wydajnościowych). Tematy poruszane w tej informacji mogą się znacznie różnić w zależności od tego, do jakiego rodzaju interesariuszy są one kierowane - czy są to interesariusze biznesowi / użytkownicy, czy też techniczni / operacyjni.

**Interesariusze zorientowani na biznes**

Podczas komunikowania się z interesariuszami biznesowymi należy wziąć pod uwagę następujące czynniki:

- Interesariusze biznesowi są mniej zainteresowani rozróżnieniem między funkcjonalnymi a niefunkcjonalnymi cechami jakości.
- Zagadnienia techniczne dotyczące narzędzi, skryptów i generowania obciążenia mają na ogół drugorzędne znaczenie.
- Należy jasno określić związek między ryzykiem produktowym a celami testów wydajnościowych.
- Zainteresowane strony muszą być świadome równowagi między kosztem planowanych testów wydajności a tym, jak reprezentatywne będą wyniki testów wydajnościowych w porównaniu z warunkami produkcji.
- Powtarzalność planowanych testów wydajnościowych musi być zakomunikowana. Czy test będzie trudny do powtórzenia, czy można go powtórzyć przy minimalnym wysiłku?
- Należy poinformować o ryzykach związanych z projektem. Obejmują one ograniczenia i zależności dotyczące konfiguracji testów, wymagań infrastrukturalnych (np. sprzęt, narzędzia, dane, przepustowość, środowiska testowe, zasoby) oraz zależności od kluczowych pracowników.
- Działania na wysokim szczeblu muszą być komunikowane (patrz podrozdziały 4.2. i 4.3.) wraz z szerokim planem zawierającym opis kosztów, harmonogramu i kamieni milowych.

**Interesariusze zorientowani na technologię**

Podczas komunikacji z zainteresowanymi stronami specjalizującymi się w technologii należy wziąć pod uwagę następujące czynniki:

- Należy wyjaśnić planowane podejście do generowania wymaganych profili obciążeń i jasno określić oczekiwane zaangażowanie interesariuszy technicznych.
- Należy wyjaśnić szczegółowe etapy konfiguracji i wykonywania testów wydajności, aby pokazać związek testowania z ryzykiem architektonicznym.
- Kroki wymagane do zapewnienia powtarzalności testów wydajności muszą zostać zakomunikowane. Mogą to być aspekty organizacyjne (np. udział kluczowego personelu), a także zagadnienia techniczne.
- W przypadku współdzielenia środowisk testowych należy przekazać harmonogram testów wydajnościowych, aby zapewnić, że nie wpłynie to negatywnie na wyniki testów.
- Należy zakomunikować i zaakceptować ograniczenia potencjalnego wpływu na rzeczywistych użytkowników, jeśli testy wydajnościowe mają być wykonane w środowisku produkcyjnym.
- Interesariusze techniczni muszą mieć jasność co do swoich zadań i harmonogramu.

## 4.2. Analiza, projektowanie i wdrażanie

### 4.2.1. Typowe protokoły komunikacyjne

Protokoły komunikacyjne definiują zestaw reguł komunikacji między komputerami a systemami. Właściwe projektowanie testów pod kątem określonych części systemu wymaga zrozumienia protokołów.

Protokoły komunikacyjne są często opisywane przez warstwy modelu OSI (ang. *Open Systems Interconnection*; patrz ISO/IEC 7498-1), chociaż niektóre protokoły mogą wykraczać poza ten model. Do testowania wydajnościowego najczęściej używane są protokoły od warstwy 5 (warstwa sesji) do warstwy 7 (warstwa aplikacji). Typowe protokoły obejmują:

- bazę danych - ODBC, JDBC, inne protokoły specyficzne dla dostawcy,
- sieć - HTTP, HTTPS, HTML,
- usługi sieciowe (ang. *Web Services*) - SOAP, REST.

Ogólnie rzecz biorąc, poziom warstwy OSI, na którym koncentruje się najwięcej testów wydajnościowych, odnosi się do poziomu testowanej architektury. Na przykład podczas testowania niektórych niskopoziomowych, wbudowanych architektur, w centrum uwagi będą głównie warstwy modelu OSI na niższych poziomach.

Dodatkowe protokoły używane w testach wydajnościowych obejmują:

- sieć - DNS, FTP, IMAP, LDAP, POP3, SMTP, Windows Sockets, CORBA,
- sieci komórkowe - TruClient, SMP, MMS,
- dostęp zdalny - Citrix ICA, RTE,
- SOA - MQSeries, JSON, WSCL.

Ważne jest, aby zrozumieć ogólną architekturę systemu, ponieważ testy wydajnościowe można wykonać na pojedynczym komponencie systemu (np. serwerze WWW, serwerze bazy danych) lub na całym systemie za pomocą testów typu *end-to-end* (testów kompleksowych).

Tradycyjne aplikacje dwuwarstwowe zbudowane w modelu klient-serwer określają „klienta” jako graficzny interfejs użytkownika i podstawowy interfejs użytkownika, a „serwer” jako bazę danych w “backendzie”. Aplikacje te wymagają użycia protokołu takiego jak ODBC w celu uzyskania dostępu do bazy danych. Wraz z ewolucją aplikacji internetowych i architektur wielopoziomowych wiele serwerów bierze udział w przetwarzaniu informacji, które są ostatecznie renderowane w przeglądarce użytkownika.

W zależności od części systemu, która ma być testowana, wymagana jest znajomość odpowiedniego protokołu, który ma być używany. Dlatego jeśli zachodzi potrzeba przeprowadzenia testów kompleksowych emulujących aktywność użytkownika z poziomu przeglądarki, zastosowany zostanie protokół sieciowy taki jak HTTP/HTTPS. W ten sposób można ominąć interakcję z GUI, a testy mogą skupić się na działaniach i komunikacji przy pomocy serwerów backend’owych.

### 4.2.2. Transakcje

Transakcje opisują zestaw czynności wykonywanych przez system od momentu zainicjowania do zakończenia jednego lub większej liczby procesów (żądań, operacji lub procesów operacyjnych). Czas reakcji (odpowiedzi) transakcji można mierzyć w celu oceny wydajności systemu. Podczas testu wydajnościowego pomiary te służą do identyfikacji wszelkich komponentów, które wymagają korekty lub optymalizacji.

Symulowane transakcje mogą obejmować “czas do namysłu”, aby lepiej odzwierciedlić czas, w którym rzeczywisty użytkownik podejmuje działanie (np. naciśnięcie przycisku „Wyślij”). Czas reakcji (odpowiedzi) transakcji plus czas do namysłu równa się czasowi, jaki upłynął dla tej transakcji.

Czasy reakcji (odpowiedzi) transakcji zebrane podczas testu wydajnościowego pokazują, jak ten pomiar zmienia się pod wpływem różnych obciążeń systemu. Analiza może wykazać brak degradacji pod obciążeniem, podczas gdy inne pomiary mogą wykazać poważną degradację. Zwiększając obciążenie i mierząc podstawowe czasy transakcji, można skorelować przyczynę degradacji z czasami odpowiedzi jednej lub więcej transakcji.

Transakcje można również zagnieżdżać, aby można było mierzyć czynności indywidualne i zagregowane. Może to być pomocne na przykład przy zrozumieniu wydajności systemu zamówień online. Tester może chcieć zmierzyć odrębne kroki w procesie zamówienia (np. wyszukiwanie pozycji, dodanie pozycji do koszyka, zapłata za przedmiot, potwierdzenie zamówienia), a także grupując transakcje w całym procesie zamówienia można zebrać oba zestawy informacji w jednym teście.

### 4.2.3. Identyfikowanie profili operacyjnych

Profile operacyjne określają różne wzorce interakcji z aplikacją, takie jak użytkownicy lub inne komponenty systemu. Dla danej aplikacji można określić wiele profili operacyjnych. Można je łączyć, tworząc pożądany profil obciążenia, aby osiągnąć określone cele testu wydajnościowego (patrz sekcja 4.2.4.).

W celu identyfikacji profili operacyjnych w tej sekcji opisano następujące podstawowe kroki:
1. Określenie danych, które mają być zebrane.
2. Zebranie danych przy wykorzystaniu jednego lub kilku źródeł.
3. Ocena danych w celu skonstruowania profili operacyjnych.

**Identyfikacja danych**

W przypadku interakcji użytkowników z testowanym systemem w celu modelowania ich profili operacyjnych (tj. sposobu ich interakcji z systemem) gromadzone lub szacowane są następujące dane:

- Różne typy person użytkowników i ich role (np. użytkownik standardowy, użytkownik zarejestrowany, administrator, grupy użytkowników z określonymi uprawnieniami).
- Różne ogólne zadania wykonywane przez tych użytkowników/role (np. przeglądanie witryny internetowej w poszukiwaniu informacji, wyszukiwanie w witrynie internetowej określonego produktu, wykonywanie czynności związanych z rolami). Należy zwrócić uwagę, że te zadania najlepiej jest modelować na wysokim poziomie abstrakcji (np. na poziomie procesów biznesowych lub głównych historyjek użytkownika).
- Szacunkowa liczba użytkowników dla każdej roli/zadania na jednostkę czasu w danym okresie. Informacje te będą również przydatne przy późniejszym budowaniu profili obciążenia (patrz sekcja 4.2.4.).

### Zbieranie danych

Powyższe dane można pozyskać z wielu różnych źródeł:

- Prowadzenie wywiadów lub warsztatów z interesariuszami, takimi jak właściciele produktów, kierownicy sprzedaży i (potencjalni) użytkownicy końcowi. Dyskusje te często ujawniają główne profile operacyjne użytkowników i dostarczają odpowiedzi na podstawowe pytanie: „Dla kogo jest przeznaczona ta aplikacja?”.
- Specyfikacje i wymagania funkcjonalne (jeśli są dostępne) są cennym źródłem informacji o planowanych wzorcach użytkowania, które mogą również pomóc w identyfikacji typów użytkowników i ich profili operacyjnych. Tam, gdzie specyfikacje funkcjonalne są wyrażone jako historyjki użytkownika, standardowy format bezpośrednio umożliwia identyfikację typów użytkowników (tj. jako <typ użytkownika> chcę <pewna możliwość>, aby <jakaś korzyść>). Podobnie diagramy i opisy przypadków użycia UML określają „aktora” przypadku użycia.
- Ocena danych użytkowania i metryk uzyskanych z podobnych aplikacji może pomóc w identyfikacji typów użytkowników i dostarczyć pewnych wstępnych wskazówek dotyczących spodziewanej liczby użytkowników. Zalecany jest dostęp do automatycznie monitorowanych danych (np. z narzędzia administracyjnego webmastera). Będzie to obejmować logi monitorowania i dane pobrane podczas użytkowania obecnego systemu operacyjnego, w przypadku którego planowana jest aktualizacja tego systemu.
- Monitorowanie zachowania użytkowników podczas wykonywania predefiniowanych zadań z aplikacją może dać wgląd w typy profili operacyjnych, które mają być modelowane do testowania wydajnościowego. Zalecane jest skoordynowanie tego zadania z planowanymi testami użyteczności (zwłaszcza jeśli dostępne jest laboratorium użyteczności).

### Utworzenie profili operacyjnych

Aby zidentyfikować i stworzyć profile operacyjne dla użytkowników, należy wykonać następujące kroki:

- Przyjmuje się podejście odgórne. Początkowo tworzone są stosunkowo proste, szerokie profile operacyjne, które są następnie dzielone tylko wtedy, gdy jest to konieczne do osiągnięcia celów testu wydajnościowego (patrz sekcja 4.1.1.).
- Poszczególne profile użytkowników mogą być wyodrębnione jako istotne dla testowania wydajnościowego, jeśli obejmują zadania, które są wykonywane często, wymagają krytycznych (wysokiego ryzyka) lub częstych transakcji między różnymi komponentami systemu lub potencjalnie wymagają przesłania dużych ilości danych.
- Profile operacyjne są przeglądane i dopracowywane z głównymi interesariuszami przed ich wykorzystaniem do tworzenia profili obciążeń (patrz sekcja 4.2.4.).

Testowany system nie zawsze jest poddawany obciążeniom ze strony użytkownika. Profile operacyjne mogą być również wymagane do testowania wydajnościowego następujących typów systemów (należy zauważyć, że ta lista nie jest wyczerpująca):

### Systemy przetwarzania wsadowego off-line

Koncentrujemy się tutaj głównie na przepustowości systemu przetwarzania wsadowego (patrz sekcja 4.2.5.) i jego zdolności do ukończenia w danym okresie. Profile operacyjne koncentrują się na rodzajach przetwarzania, które są wymagane w procesach wsadowych. Na przykład profile operacyjne systemu obrotu akcjami (który zazwyczaj obejmuje przetwarzanie transakcji online i partiami) mogą obejmować profile dotyczące transakcji płatniczych, weryfikacji danych uwierzytelniających i sprawdzania zgodności warunków prawnych dla określonych rodzajów transakcji giełdowych. Każdy z tych profili operacyjnych skutkowałby różnymi ścieżkami przechodzenia przez cały proces wsadowy dla zapasów. Kroki opisane powyżej w celu identyfikacji profili operacyjnych systemów online opartych na użytkownikach można również zastosować w kontekście przetwarzania wsadowego.

**Systemy systemów**

Komponenty w środowisku wielosystemowym (które mogą być również wbudowane) odpowiadają różnym typom danych wejściowych z innych systemów lub komponentów. W zależności od charakteru testowanego systemu może to wymagać modelowania kilku różnych profili operacyjnych, aby w sposób skuteczny reprezentować rodzaje danych wejściowych dostarczanych przez te zewnętrzne systemy. Może to obejmować szczegółową analizę (np. buforów i kolejek) wspólnie z architektami systemu w oparciu o specyfikacje systemu i interfejsu.

## 4.2.4. Tworzenie profili obciążenia

Profil obciążenia określa aktywność, której może doświadczyć testowany komponent lub system podczas produkcji. Składa się z określonej liczby instancji, które będą wykonywać działania z predefiniowanych profili operacyjnych w określonym przedziale czasu. W przypadku użytkowników, powszechnie stosowany jest termin „użytkownicy wirtualni”.

Podstawowe informacje wymagane do stworzenia realistycznego i powtarzalnego profilu obciążenia to:

- cel testowania wydajnościowego (np. ocena zachowania systemu pod obciążeniem),
- profile operacyjne, które dokładnie reprezentują indywidualne wzorce użytkowania (patrz sekcja 4.2.3.),
- znane problemy z przepustowością i współbieżnością (patrz sekcja 4.2.5.),
- ilość i rozkład czasu, z jakim profile operacyjne mają być uruchomione tak, aby testowany system doświadczył żądanego obciążenia. Typowe przykłady to:
    - przyspieszenie (ang. *ramp-up*): stale rosnące obciążenie (np. dodawanie jednego wirtualnego użytkownika na minutę),
    - spowolnienie (ang. *ramp-down*): stale zmniejszające się obciążenie,
    - skoki: natychmiastowe zmiany obciążenia (np. dodaj 100 wirtualnych użytkowników co pięć minut),
    - predefiniowane rozkłady (np. wolumen imituje dzienne lub okresowe cykle biznesowe).

Poniższy przykład przedstawia konstrukcję profilu obciążenia w celu wygenerowania warunków przeciążenia (na poziomie lub powyżej oczekiwanego maksimum dla systemu) dla testowanego systemu.

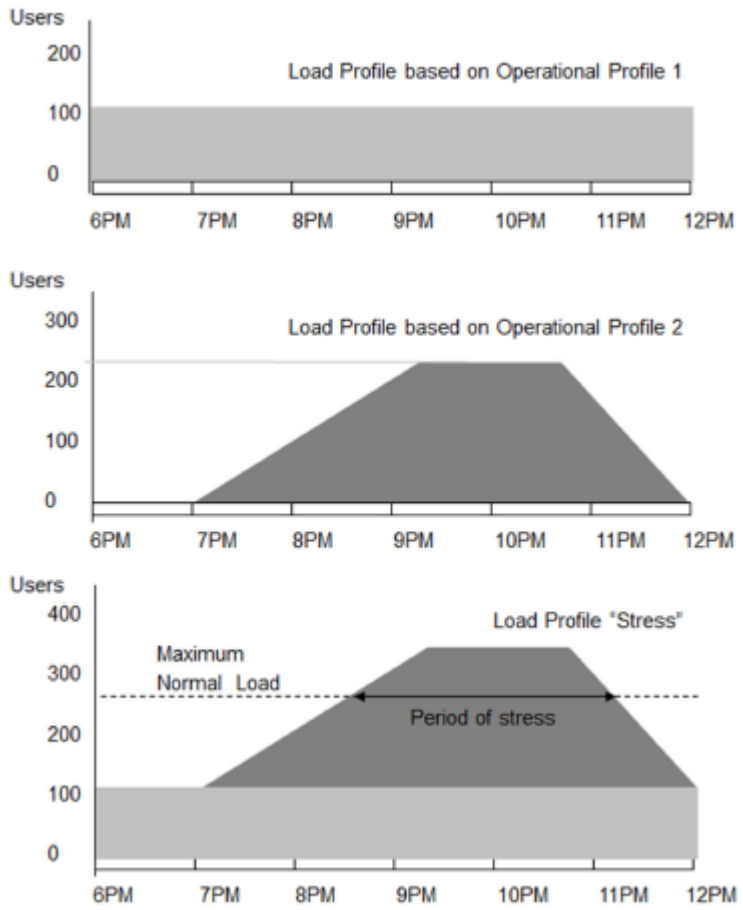


Diagram 1. Przykład konstruowania “przeciążeniowego” profilu obciążenia

W górnej części diagramu pokazano profil obciążenia, który składa się ze skokowego wprowadzenia 100 wirtualnych użytkowników. Użytkownicy ci wykonują czynności określone przez Profil Operacyjny 1 przez cały czas trwania testu. Jest to typowe dla wielu profili obciążenia wydajności, które reprezentują obciążenie w tle.

Środkowy wykres przedstawia profil obciążenia, który składa się z przyspieszenia (ang. *ramp-up*) do 220 wirtualnych użytkowników, który jest utrzymywany przez dwie godziny, aż do spowolnienia (ang. *ramp-down*). Każdy użytkownik wirtualny wykonuje czynności określone w Profilu Operacyjnym 2.

Dolny wykres przedstawia profil obciążenia, który wynika z kombinacji dwóch opisanych powyżej. Testowany system poddawany jest trzygodzinnym obciążeniom. Dalsze przykłady można znaleźć w [Bath14].

### 4.2.5. Analiza przepustowości i współbieżności

Ważnym jest, aby zrozumieć różne aspekty obciążenia: przepustowość i współbieżność. Aby prawidłowo modelować profile operacyjne i obciążenia, należy wziąć pod uwagę oba aspekty.

**Przepustowość systemu**

Przepustowość systemu jest miarą liczby transakcji danego typu, które system przetwarza w jednostce czasu. Na przykład liczba zamówień na godzinę lub liczba żądań HTTP na sekundę. Przepustowość systemu należy odróżnić od przepustowości sieci, czyli ilości danych przesyłanych przez sieć (patrz podrozdział 2.1.).

Przepustowość systemu określa stopień obciążenia systemu. Niestety, dość często liczba współbieżnych użytkowników jest używana do definiowania obciążenia dla systemów interaktywnych zamiast przepustowości. Jest to częściowo prawdą, ponieważ ta liczba jest często łatwiejsza do znalezienia, a częściowo dlatego, że jest to sposób, w jaki narzędzia do testowania obciążenia definiują obciążenie. Bez definiowania profili operacyjnych - co każdy użytkownik robi i jak intensywnie (co również oznacza przepustowość dla jednego użytkownika) - liczba użytkowników nie jest dobrą miarą obciążenia. Na przykład, jeśli 500 użytkowników wykonuje krótkie zapytania w ciągu minuty, mamy przepustowość 30 000 zapytań na godzinę. Jeśli tych samych 500 użytkowników uruchamia te same zapytania, ale jedno na godzinę, przepustowość wynosi 500 zapytań na godzinę. Jest więc tych samych 500 użytkowników, ale 60-krotna różnica między obciążeniami i co najmniej 60-krotna różnica w wymaganiach sprzętowych systemu.

Modelowanie obciążenia zwykle odbywa się z uwzględnieniem liczby użytkowników wirtualnych (wątków wykonawczych) i czasu do namysłu (opóźnienia między działaniami użytkownika). Jednak przepustowość systemu jest również określana przez czas przetwarzania, który może wzrosnąć wraz ze wzrostem obciążenia.

Przepustowość systemu = [liczba wirtualnych użytkowników] / ([czas przetwarzania] + [czas do namysłu])

Kiedy więc czas przetwarzania rośnie, przepustowość może znacznie spaść, nawet jeśli wszystko inne pozostaje bez zmian.

Przepustowość systemu jest ważnym aspektem podczas testowania systemów przetwarzania wsadowego. W tym przypadku przepustowość jest zwykle mierzona na podstawie liczby transakcji, które można zrealizować w danym przedziale czasu (np. nocne okno przetwarzania wsadowego).

**Współbieżność**

Współbieżność jest miarą liczby jednoczesnych / równolegle wykonywanych wątków. W przypadku systemów interaktywnych może to być wielu jednoczesnych / równoległych użytkowników. Współbieżność jest zwykle modelowana w narzędziach do testowania obciążenia przez ustawienie liczby wirtualnych użytkowników.

Współbieżność jest ważną miarą. Reprezentuje liczbę sesji równoległych, z których każda może korzystać z własnych zasobów. Nawet jeśli przepustowość jest taka sama, ilość używanych zasobów może się różnić w zależności od współbieżności. Typowe konfiguracje testów to systemy zamknięte (z punktu widzenia teorii kolejek), w których ustalana jest liczba użytkowników w systemie (stała populacja). Jeśli wszyscy użytkownicy czekają na odpowiedź systemu w systemie zamkniętym, nowi użytkownicy nie mogą się do niego dostać. Wiele systemów publicznych to systemy otwarte - przez cały czas pojawiają się nowi użytkownicy, nawet jeśli wszyscy obecni użytkownicy czekają na odpowiedź systemu.

### 4.2.6. Podstawowa struktura skryptu testu wydajnościowego

Skrypt testu wydajnościowego powinien symulować działanie użytkownika lub komponentu, które przyczynia się do obciążenia testowanego systemu (którym może być cały system lub jeden z jego składników). Inicjuje zapytania do serwera w odpowiedniej kolejności i w zadanym tempie.

Najlepszy sposób tworzenia skryptów testów wydajnościowych zależy od zastosowanego podejścia do generowania obciążenia (patrz podrozdział 4.1.).

- Tradycyjnym sposobem jest rejestrowanie komunikacji między klientem a systemem lub komponentem na poziomie protokołu, a następnie odtwarzanie jej po sparametryzowaniu i udokumentowaniu skryptu. W wyniku parametryzacji powstaje skalowalny i możliwy do utrzymania skrypt, ale zadanie parametryzacji może być czasochłonne.
- Rejestrowanie na poziomie GUI zazwyczaj obejmuje przechwytywanie działań GUI pojedynczego klienta za pomocą narzędzia do wykonywania testów i uruchamianie tego skryptu za pomocą narzędzia do generowania obciążenia w celu reprezentowania wielu klientów.
- Programowanie może odbywać się przy użyciu żądań protokołów (np. żądań HTTP), akcji GUI lub wywołań API. W przypadku skryptów programistycznych należy określić dokładną kolejność żądań wysyłanych i odbieranych z rzeczywistego systemu, co może nie być trywialne.

Zwykle skrypt to jedna lub kilka sekcji kodu (napisanych w generycznym języku programowania z pewnymi rozszerzeniami lub w języku specjalistycznym) lub obiekt, który może być przedstawiony użytkownikowi przez narzędzie w graficznym interfejsie użytkownika. W obu przypadkach skrypt będzie zawierał żądania serwera generujące obciążenie (np. żądania HTTP) i pewną logikę programowania wokół nich określającą, jak dokładnie te żądania będą wywoływane (np. w jakiej kolejności, w jakim momencie, z jakimi parametrami, co należy sprawdzić). Im bardziej wyrafinowana logika, tym większa potrzeba używania języków programowania o dużych możliwościach.

### Ogólna struktura

Często skrypt zawiera sekcję inicjalizacyjną (w której wszystko jest przygotowywane dla głównej części), sekcje główne, które mogą być wykonywane wielokrotnie oraz sekcję porządkującą (w której są podejmowane niezbędne kroki, aby poprawnie zakończyć test).

### Gromadzenie danych

Aby zebrać czasy odpowiedzi, do skryptu należy dodać liczniki czasu, które mierzą, jak długo trwa wykonywanie żądania lub kombinacji żądań. Żądania czasowe powinny odpowiadać znaczącej jednostce pracy logicznej - na przykład transakcji biznesowej polegającej na dodaniu pozycji do zamówienia lub złożeniu zamówienia.

Ważne jest, aby zrozumieć, co dokładnie jest mierzone: w przypadku skryptów na poziomie protokołu jest to tylko czas odpowiedzi serwera i sieci, podczas gdy skrypty GUI mierzą czasy całkowite (chociaż to, co dokładnie jest mierzone, zależy od zastosowanej technologii).

### Weryfikacja wyników i obsługa błędów

Ważną częścią skryptu jest weryfikacja wyników i obsługa błędów. Nawet w najlepszych narzędziach do testowania obciążenia domyślna obsługa błędów jest zwykle minimalna (na przykład sprawdzanie kodu zwrotnego żądania HTTP), dlatego zaleca się dodanie dodatkowych sprawdzeń, aby zweryfikować, co faktycznie zwracają żądania. Również jeśli w przypadku błędu wymagane jest jakiekolwiek czyszczenie, prawdopodobnie będzie trzeba je wdrożyć ręcznie. Dobrą praktyką jest sprawdzenie, czy skrypt robi to, co powinien, przy użyciu metod pośrednich - na przykład sprawdzenie bazy danych w celu weryfikacji, czy zostały dodane odpowiednie informacje.

Skrypty mogą zawierać inną logikę określającą reguły dotyczące czasu i sposobu wysyłania żądań do serwera. Jednym z przykładów jest ustawienie punktów synchronizacji, co jest wykonywane przez określenie, że skrypt powinien czekać na zdarzenie w tym punkcie, zanim będzie kontynuował. Punkty synchronizacji mogą służyć do zapewnienia jednoczesnego wywołania określonej akcji lub do koordynowania pracy między kilkoma skryptami. Skrypty do testowania wydajnościowego to oprogramowanie, więc tworzenie skryptu do testowania wydajnościowego to czynność programistyczna. Powinien obejmować zapewnienie jakości i testy sprawdzające w celu weryfikacji, czy skrypt działa zgodnie z oczekiwaniami z całym zakresem danych wejściowych.

## 4.2.7. Wdrażanie skryptów testów wydajnościowych

Skrypty testów wydajnościowych są wdrażane w oparciu o PTW (Plan Testów Wydajnościowych) i profile obciążenia. Chociaż szczegóły techniczne wdrożenia będą się różnić w zależności od podejścia i zastosowanych narzędzi, cały proces pozostaje taki sam. Skrypt wydajności jest tworzony przy użyciu zintegrowanego środowiska programistycznego (IDE) lub edytora skryptów w celu symulacji zachowania użytkownika lub komponentu. Zwykle skrypt jest tworzony w celu symulacji określonego profilu operacyjnego (chociaż często można połączyć kilka profili operacyjnych w jeden skrypt z instrukcjami warunkowymi).

Po ustaleniu kolejności żądań skrypt może zostać nagrany lub zaprogramowany w zależności od podejścia. Nagrywanie zazwyczaj gwarantuje, że dokładnie symuluje rzeczywisty system, podczas gdy programowanie opiera się na znajomości właściwej kolejności żądań.

Jeśli stosuje się nagrywanie na poziomie protokołu, w większości przypadków istotnym krokiem po nagraniu jest zastąpienie wszystkich zarejestrowanych identyfikatorów wewnętrznych, które definiują kontekst. Identyfikatory te muszą być przekształcone w zmienne, które można zmieniać między uruchomieniami, z odpowiednimi wartościami, które są wyodrębniane z odpowiedzi na żądanie (np. identyfikator użytkownika uzyskiwany podczas logowania musi być podawany dla wszystkich kolejnych transakcji). Jest to część parametryzacji skryptu, czasami nazywana „korelacją". W tym kontekście słowo korelacja ma inne znaczenie niż w statystykach (gdzie oznacza związek między dwiema lub więcej rzeczami). Zaawansowane narzędzia do testowania obciążenia mogą automatycznie dokonywać niektórych korelacji, więc w niektórych przypadkach może być ona transparentna, ale w bardziej złożonych przypadkach może być wymagana ręczna korelacja lub dodanie nowych reguł korelacji. Nieprawidłowa korelacja lub brak korelacji jest głównym powodem niepowodzenia odtwarzania nagranych skryptów.

Uruchamianie wielu wirtualnych użytkowników z tą samą nazwą użytkownika i uzyskiwanie dostępu do tego samego zestawu danych (co zwykle ma miejsce podczas odtwarzania nagranego skryptu bez żadnych dalszych modyfikacji poza konieczną korelacją) to prosty sposób na uzyskanie mylących wyników. Dane mogłyby być całkowicie buforowane (kopiowane z dysku do pamięci w celu szybszego dostępu), a wyniki byłyby znacznie lepsze niż na produkcji (gdzie takie dane można odczytać z dysku). Korzystanie z tych samych użytkowników i/lub danych może również powodować problemy ze współbieżnością (np. jeśli dane są zablokowane, gdy użytkownik je aktualizuje), a wyniki byłyby znacznie gorsze niż w środowisku produkcyjnym, ponieważ oprogramowanie czekałoby na zwolnienie blokady przed następnym użytkownikiem, który może zablokować dane do aktualizacji.

Dlatego skrypty i jarzmo testowe powinny być sparametryzowane (tj. ustalone lub zarejestrowane dane należy zastąpić wartościami z listy możliwych wyborów), tak, aby każdy wirtualny użytkownik korzystał z odpowiedniego zestawu danych. Termin „właściwy" oznacza tutaj na tyle inny, że pozwala uniknąć problemów z buforowaniem i współbieżnością, które są specyficzne dla systemu, danych i wymagań testowych. Ta dalsza parametryzacja zależy od danych w systemie i sposobu, w jaki system pracuje z tymi danymi, więc zwykle odbywa się to ręcznie, chociaż wiele narzędzi zapewnia tutaj pomoc.

Istnieją przypadki, w których niektóre dane muszą zostać sparametryzowane, aby test działał więcej niż raz - na przykład, gdy tworzone jest zamówienie, a nazwa zamówienia musi być unikalna. Dopóki nazwa zamówienia nie będzie sparametryzowana, test zakończy się niepowodzeniem, gdy tylko spróbuje utworzyć zamówienie z istniejącą (zarejestrowaną) nazwą.

Aby dopasować profile operacyjne, należy wstawić i/lub skorygować (jeśli zostały zarejestrowane) czasy do namysłu, aby wygenerować odpowiednią liczbę żądań/przepustowość, jak omówiono w sekcji 4.2.5.

Gdy tworzone są skrypty dla oddzielnych profili operacyjnych, są one łączone w scenariusz implementujący cały profil obciążenia. Profil obciążenia kontroluje, ilu wirtualnych użytkowników jest uruchamianych przy użyciu każdego skryptu, kiedy i z jakimi parametrami. Szczegóły implementacji zależą od konkretnego narzędzia do testowania obciążenia lub wiązki przewodów.

### 4.2.8. Przygotowanie do wykonania testu wydajnościowego

Do głównych czynności przygotowujących do wykonania testów wydajnościowych należą:

- konfiguracja testowanego systemu,
- wdrażanie środowiska,
- konfiguracja narzędzi do generowania i monitorowania obciążenia oraz upewnienie się, że zostaną zebrane wszystkie niezbędne informacje.

Ważne jest, aby środowisko testowe było jak najbardziej zbliżone do środowiska produkcyjnego. Jeśli nie jest to możliwe, należy dokładnie zrozumieć różnice i sposób, w jaki wyniki testów będą odwzorowane w środowisku produkcyjnym. Idealnie byłoby, gdyby używane było prawdziwe środowisko produkcyjne i dane, ale testowanie w środowisku skalowalnym nadal może pomóc złagodzić szereg zagrożeń związanych z wydajnością.

Należy pamiętać, że wydajność jest nieliniową funkcją środowiska, więc im mniej zbliżone jest środowisko testowe do standardu produkcyjnego, tym trudniej jest sporządzić dokładne prognozy wydajności produkcji. Brak wiarygodności prognoz i podwyższony poziom ryzyka rosną w miarę, jak system testowy mniej przypomina system produkcyjny

Najważniejszymi częściami środowiska testowego są dane, konfiguracja sprzętu i oprogramowania oraz konfiguracja sieci. Rozmiar i struktura danych może znacząco wpłynąć na wyniki testów obciążenia. Korzystanie z małego zestawu próbek danych lub zestawu próbek o różnej złożoności danych do testów wydajnościowych może dać mylące wyniki, zwłaszcza gdy system produkcyjny będzie używał dużego zestawu danych. Trudno jest przewidzieć, w jakim stopniu rozmiar danych wpłynie na wydajność przed wykonaniem rzeczywistych testów. Im bardziej dane testowe są zbliżone do danych produkcyjnych pod względem wielkości i struktury, tym bardziej wiarygodne będą wyniki testów.

Jeśli dane są generowane lub zmieniane podczas testu, może być konieczne przywrócenie oryginalnych danych przed następnym cyklem testowym, aby upewnić się, że system jest we właściwym stanie.

Jeśli niektóre części systemu lub niektóre dane są z jakiegoś powodu niedostępne do testów wydajnościowych, należy obejść ten problem. Na przykład można zaimplementować kod pośredniczący, aby zastąpić i emulować komponent strony trzeciej odpowiedzialny za przetwarzanie karty kredytowej. Ten proces jest często określany jako „wirtualizacja usług”

i dostępne są specjalne narzędzia wspomagające ten proces. Zdecydowanie zaleca się użycie takich narzędzi w celu odizolowania testowanego systemu.

Istnieje wiele sposobów wdrażania środowisk. Przykładowe możliwości wdrożenia środowiska testowego są następujące:

- tradycyjne wewnętrzne (i zewnętrzne) laboratoria testowe,
- chmura jako środowisko wykorzystujące infrastrukturę jako usługę (ang. *Infrastructure as a Service* - IaaS), gdy niektóre części systemu lub cały system są wdrażane w chmurze,
- chmura jako środowisko korzystające z oprogramowania jako usługi (ang. *Software as a Service* - SaaS), gdy dostawcy świadczą usługę testowania obciążenia.

W zależności od konkretnych celów i systemów do przetestowania, dane środowisko testowe może być bardziej preferowane niż inne, na przykład:

- Aby przetestować efekt poprawy wydajności (optymalizacji wydajności), użycie izolowanego środowiska laboratoryjnego może być lepszą opcją, aby zobaczyć nawet niewielkie zmiany wprowadzone jako rezultat modyfikacji.
- Aby przetestować całe środowisko produkcyjne kompleksowo (*end-to-end)* i upewnić się, że system poradzi sobie z obciążeniem bez większych problemów, bardziej odpowiednie może być testowanie z chmury lub usługi. Warto zwrócić uwagę, że działa to tylko w przypadku systemów, do których istnieje dostęp z chmury.
- Aby zminimalizować koszty, gdy testowanie wydajnościowe jest ograniczone w czasie, bardziej ekonomicznym rozwiązaniem może być utworzenie środowiska testowego w chmurze.

Niezależnie od zastosowanego podejścia do wdrożenia, sprzęt i oprogramowanie należy skonfigurować tak, aby spełniały cel i plan testu. Jeśli środowisko testowe odpowiada środowisku produkcyjnemu, należy je skonfigurować w ten sam sposób. Jeśli jednak istnieją różnice, może być konieczne dostosowanie konfiguracji, aby uwzględnić te różnice. Na przykład, jeśli maszyny testowe mają mniej pamięci fizycznej niż maszyny produkcyjne, może zajść potrzeba dostosowania parametrów pamięci oprogramowania (takich jak rozmiar sterty Java), aby uniknąć stronicowania pamięci.

Właściwa konfiguracja / emulacja sieci jest ważna dla systemów globalnych i mobilnych. W przypadku systemów globalnych (tj. takich, które mają użytkowników na całym świecie lub przetwarzanie jest rozproszone) jednym z podejść może być rozmieszczenie generatorów obciążenia w miejscach, w których znajdują się użytkownicy. W przypadku systemów mobilnych emulacja sieci pozostaje najbardziej realną opcją ze względu na różnice w typach sieci, których można użyć. Niektóre narzędzia do testowania obciążenia mają wbudowane narzędzia do emulacji sieci. Istnieją też niezależne narzędzia do emulacji sieci.

Narzędzia do generowania obciążenia powinny być prawidłowo wdrożone, a narzędzia monitorujące powinny być skonfigurowane tak, aby zbierać wszystkie niezbędne metryki testu. Lista metryk zależy od celów testów, ale zaleca się zebranie przynajmniej podstawowych metryk dla wszystkich testów (patrz sekcja 2.1.2.).

W zależności od obciążenia, określonego narzędzia / podejścia do generowania obciążenia i konfiguracji maszyny może być potrzebnych więcej niż jedna maszyna do generowania obciążenia. Aby zweryfikować konfigurację, należy monitorować również maszyny biorące udział w generowaniu obciążenia. Pomoże to uniknąć sytuacji, w której obciążenie nie jest odpowiednio utrzymywane, ponieważ jeden z generatorów obciążenia działa wolno.

W zależności od konfiguracji i używanych narzędzi, narzędzia do testowania obciążenia muszą być skonfigurowane w celu utworzenia odpowiedniego obciążenia. Na przykład można ustawić określone parametry emulacji przeglądarki lub wykorzystać fałszowanie adresu IP (*IP spoofing*) symulując, że każdy użytkownik wirtualny ma inny adres IP.

Przed wykonaniem testów należy zweryfikować środowisko i konfigurację. Odbywa się to zwykle poprzez przeprowadzenie kontrolowanego zestawu testów i weryfikację wyników testów, a także sprawdzenie, czy narzędzia monitorujące śledzą istotne informacje.

Aby sprawdzić, czy test działa zgodnie z założeniami, można zastosować różne techniki, w tym analizę logów i weryfikację zawartości bazy danych. Przygotowanie do testu obejmuje sprawdzenie, czy wymagane informacje zostały zarejestrowane, system jest w odpowiednim stanie itp. Na przykład, jeśli test znacząco zmieni stan systemu (dodanie / zmiana informacji w bazie danych), może być konieczne przywrócenie systemu do stanu pierwotnego przed powtórzeniem testu.

## 4.3. Wykonywanie testów

Wykonywanie testów wydajnościowych obejmuje generowanie obciążenia dla testowanego systemu zgodnie z profilem obciążenia (zwykle realizowane przez skrypty testujące wydajność uruchamiane zgodnie z danym scenariuszem), monitorowanie wszystkich części środowiska oraz gromadzenie i przechowywanie wszystkich wyników i informacji związanych z testem. Zwykle zaawansowane narzędzia / jarzma testowe wykonują te zadania automatycznie (oczywiście po odpowiedniej konfiguracji). Zwykle też zapewniają konsolę umożliwiającą monitorowanie danych dotyczących wydajności podczas testu i umożliwiającą wprowadzenie niezbędnych korekt (patrz podrozdział 5.1.). Jednak w zależności od używanego narzędzia, testowanego systemu i konkretnych wykonywanych testów mogą być potrzebne pewne czynności wykonywane ręcznie.

Testy wydajnościowe zwykle wykonywane są wtedy, gdy stan systemu jest stabilny. Na przykład, gdy wszyscy symulowani użytkownicy / wątki są inicjowani / inicjowane i wykonują zadania zgodnie z projektem. Kiedy zmienia się obciążenie (na przykład, gdy dodaje się nowych użytkowników), zmienia się zachowanie systemu i coraz trudniej jest monitorować i analizować wyniki testów. Etap dochodzenia do stanu stabilnego jest często określany jako przyspieszenie (ang. *ramp-up*), a etap kończenia testu jest często określany jako spowolnienie (ang. *ramp-down*).

Czasami ważne jest, aby przetestować stany przejściowe, gdy zmienia się zachowanie systemu. Może to dotyczyć na przykład jednoczesnego rejestrowania dużej liczby użytkowników lub testowania skokowego (ang. *spike testing*). Podczas testowania stanów przejściowych ważne jest, aby zrozumieć potrzebę uważnego monitorowania i analizy wyników, ponieważ niektóre standardowe podejścia - takie jak monitorowanie średnich - mogą być bardzo mylące.

Podczas przyspieszania zaleca się wdrażanie przyrostowych stanów obciążenia, aby monitorować wpływ stale rosnącego obciążenia na reakcję systemu. Zapewnia to wystarczającą ilość czasu na przyspieszenie oraz to, że system jest w stanie obsłużyć obciążenie. Po osiągnięciu stabilnego stanu dobrą praktyką jest monitorowanie, czy zarówno obciążenie, jak i odpowiedzi systemu, są stabilne oraz czy przypadkowe zmiany (które zawsze istnieją) nie są poważne.

Ważne jest, aby określić, jak należy obsługiwać awarie, aby mieć pewność, że nie pojawią się żadne problemy z systemem. Na przykład może być ważne, aby użytkownik wylogował się, gdy wystąpi awaria, aby upewnić się, że wszystkie zasoby powiązane z tym użytkownikiem zostały zwolnione.

Jeśli monitorowanie jest wbudowane w narzędzie do testowania obciążenia i jest odpowiednio skonfigurowane, zwykle rozpoczyna się w tym samym czasie, co wykonanie testu. Jeśli jednak używane są niezależne narzędzia monitorowania, monitorowanie należy rozpocząć oddzielnie i zebrać niezbędne informacje w taki sposób, aby można było przeprowadzić późniejszą analizę wraz z wynikami testów. To samo dotyczy analizy logów. Istotne jest zsynchronizowanie czasu wszystkich używanych narzędzi, aby można było znaleźć wszystkie informacje związane z określonym cyklem wykonywania testów.

Wykonywanie testów jest często monitorowane przy pomocy konsoli narzędzia do testów wydajnościowych i analizy logów w czasie rzeczywistym, w celu sprawdzenia problemów i błędów zarówno w teście, jak i testowanym systemie (SUT – *System Under Test*). Pomaga to uniknąć niepotrzebnego kontynuowania przeprowadzania testów na dużą skalę, które mogą nawet wpłynąć na inne systemy, jeśli coś pójdzie nie tak (np. jeśli wystąpi awaria, awaria komponentów lub generowane obciążenia są zbyt niskie lub wysokie). Testy te mogą być kosztowne do uruchomienia i może być konieczne zatrzymanie testu lub dokonanie pewnych zmian w locie w teście wydajnościowym lub konfiguracji systemu, jeśli test odbiega od oczekiwanego zachowania.

Jedną z technik weryfikowania testów obciążenia, które komunikują się bezpośrednio na poziomie protokołu, jest uruchomienie kilku skryptów funkcjonalnych na poziomie GUI lub nawet ręczne wykonanie podobnych profili operacyjnych równolegle z uruchomionym testem obciążeniowym. W ten sposób sprawdzamy, czy czasy odpowiedzi zarejestrowane podczas testu różnią się od czasów odpowiedzi mierzonych ręcznie na poziomie GUI po stronie klienta.

W niektórych przypadkach podczas przeprowadzania testów wydajnościowych w sposób zautomatyzowany (na przykład w ramach ciągłej integracji, jak omówiono w podrozdziale 3.4.), sprawdzanie musi być wykonywane automatycznie, ponieważ ręczne monitorowanie

i interwencja mogą być niemożliwe do wykonania. W takim przypadku zestaw testów powinien być w stanie rozpoznać wszelkie odchylenia lub problemy i wysłać ostrzeżenie (zwykle podczas prawidłowego zakończenia testu). Podejście to jest łatwiejsze do wdrożenia w przypadku regresyjnych testów wydajnościowych, gdy zachowanie systemu jest ogólnie znane, ale może być trudniejsze w przypadku eksploracyjnych testów wydajnościowych lub kosztownych testów wydajnościowych na dużą skalę, które mogą wymagać dynamicznych zmian w trakcie testu.

## 4.4. Analiza wyników i raportowanie

W sekcji 4.1.2. omówiono różne metryki w planie testów wydajnościowych. Zdefiniowanie ich z góry określa, co należy zmierzyć dla każdego przebiegu testu. Po zakończeniu cyklu testów należy zebrać dane dla zdefiniowanych metryk.

Podczas analizy danych najpierw porównuje się je z celem testu wydajnościowego. Po zrozumieniu zachowania można wyciągnąć wnioski, które dostarczą miarodajnego raportu podsumowującego, zawierającego zalecane działania. Działania te mogą obejmować zmianę komponentów fizycznych (np. sprzętu, routerów), zmianę oprogramowania (np. optymalizację aplikacji i wywołań bazy danych) oraz zmianę sieci (np. *load balancing, routing*).

Zazwyczaj analizowane są następujące dane:

- **Status symulowanych (np. wirtualnych) użytkowników.** To należy zbadać w pierwszej kolejności. Zwykle oczekuje się, że wszyscy symulowani użytkownicy byli w stanie wykonać zadania określone w profilu operacyjnym. Każda przerwa w tej czynności imituje to, czego może doświadczyć rzeczywisty użytkownik. Dlatego bardzo ważne jest, aby najpierw sprawdzić, czy wszystkie czynności użytkownika zostały zakończone, ponieważ wszelkie napotkane błędy mogą wpłynąć na inne dane dotyczące wydajności.

- **Czas reakcji (odpowiedzi) transakcji.** Można go mierzyć na wiele sposobów, w tym minimum, maksimum, średnią i percentyl (np. 90). Minimalne i maksymalne odczyty pokazują skrajne działanie systemu. Średnia wydajność niekoniecznie wskazuje na coś innego niż średnia matematyczna i często może być wypaczona przez wartości odstające. 90. percentyl jest często używany jako cel, ponieważ reprezentuje większość użytkowników osiągających określony próg wydajności. Nie zaleca się wymagania stuprocentowej zgodności z celami dotyczącymi wydajności, ponieważ wymagane zasoby mogą być zbyt duże, a efekt finalny dla użytkowników będzie często niewielki.

- **Transakcje na sekundę.** Dostarcza informacji o tym, ile pracy wykonał system (przepustowość systemu).

- **Niepowodzenia transakcji.** Dane te są wykorzystywane podczas analizy transakcji na sekundę. Niepowodzenia wskazują, że oczekiwane zdarzenie lub proces nie zakończyło się lub nie zostało wykonane. Wszelkie napotkane awarie są powodem do niepokoju i należy zbadać ich podstawową przyczynę (ang. *root cause*). Nieudane transakcje mogą również skutkować nieprawidłowymi danymi transakcji na sekundę, ponieważ nieudana transakcja zajmie znacznie mniej czasu niż zakończona.

- **Trafienia (lub żądania) na sekundę.** Daje to poczucie liczby trafień na serwer przez symulowanych użytkowników podczas każdej sekundy testu.

- **Przepustowość sieci.** Jest ona zwykle mierzona w bitach przez przedział czasu, przykładowo w bitach na sekundę. Stanowi to ilość danych, które symulowani użytkownicy otrzymują z serwera w każdej sekundzie (patrz sekcja 4.2.5.).

- **Odpowiedzi HTTP.** Są one mierzone na sekundę i obejmują możliwe kody odpowiedzi takie jak: 200, 302, 304, 404. Ten ostatni wskazuje, że strona nie została znaleziona.

Chociaż wiele z tych informacji można przedstawić w tabelach, prezentacje graficzne ułatwiają przeglądanie danych i identyfikację trendów.

Techniki stosowane w analizie danych mogą obejmować:
- porównanie wyników z określonymi wymaganiami,
- obserwowanie trendów w wynikach,
- statystyczne techniki kontroli jakości,
- identyfikowanie błędów,
- porównanie oczekiwanych i rzeczywistych wyników,
- porównanie wyników z wynikami poprzednich testów,
- weryfikacja poprawności działania komponentów (np. serwerów, sieci).

Identyfikacja korelacji między metrykami może pomóc nam zrozumieć, w którym momencie wydajność systemu zaczyna spadać. Na przykład, jaka liczba transakcji na sekundę była przetwarzana, gdy procesor osiągnął 90% wydajności, a system zwolnił.

Analiza może pomóc w zidentyfikowaniu podstawowej przyczyny pogorszenia wydajności lub awarii, co z kolei ułatwi naprawę. Testy potwierdzające pomogą określić, czy działanie naprawcze wyeliminowało podstawową przyczynę.

### Raportowanie

Wyniki analizy są scalane i porównywane z celami określonymi w planie testów wydajnościowych. Mogą one być podane w ogólnym raporcie o postępie testów wraz z innymi wynikami testów lub zawarte w dedykowanym raporcie dotyczącym testów wydajnościowych. Poziom szczegółowości raportu powinien odpowiadać potrzebom interesariuszy. Zalecenia oparte na tych wynikach zwykle dotyczą kryteriów wydania oprogramowania (w tym środowiska docelowego) lub wymaganych udoskonaleń wydajności.

Typowy raport z testów wydajnościowych może obejmować:

**Podsumowanie dla kierownictwa**
Ta sekcja jest wypełniana po wykonaniu wszystkich testów wydajnościowych i przeanalizowaniu i zrozumieniu wszystkich wyników. Celem jest przedstawienie zwięzłych i zrozumiałych wniosków, ustaleń i rekomendacji dla kierownictwa w celu osiągnięcia konstruktywnych rezultatów.

**Wyniki testu**
Wyniki testu mogą zawierać niektóre lub wszystkie z następujących informacji:

- Podsumowanie zawierające wyjaśnienie i omówienie wyników.
- Wyniki testu bazowego, który służy jako „migawka" (ang. *snapshot*) wydajności systemu w danym czasie i stanowi podstawę porównania z kolejnymi testami. Wyniki powinny obejmować datę/godzinę rozpoczęcia testu, cel współbieżnego użytkownika, zmierzoną przepustowość i najważniejsze ustalenia. Kluczowe ustalenia mogą obejmować zmierzony ogólny poziom błędów, czas odpowiedzi i średnią przepustowość.
- Wysokopoziomowy diagram przedstawiający komponenty architektury, które mogą (lub miały) wpływ na cele testu.
- Szczegółowa analiza (tabele i wykresy) wyników testów pokazująca czasy odpowiedzi, wskaźniki transakcji, wskaźniki błędów i analizę wydajności. Analiza zawiera również opis tego, co zaobserwowano, np. w którym momencie stabilna aplikacja stała się niestabilna oraz źródło awarii (np. serwer WWW, serwer bazy danych).

**Logi testów / zapisane informacje**
Należy rejestrować logi każdego przebiegu testowego. Logi zawierają zazwyczaj następujące informacje:

- data/godzina rozpoczęcia testu,
- czas trwania testu,
- skrypty używane do testów (w tym zbiór skryptów, jeśli używanych jest wiele skryptów) i odpowiednie dane konfiguracyjne skryptów,
- plik(i) danych testowych używane przez podczas testu,
- nazwa i lokalizacja plików danych / logów utworzonych podczas testu,
- konfiguracja sprzętu/oprogramowania (szczególnie wszelkie zmiany między przebiegami testów),
- średnie i szczytowe wykorzystanie procesora i pamięci RAM na serwerach WWW i bazodanowych,
- uwagi dotyczące osiągniętych wyników,
- zidentyfikowane usterki.

**Rekomendacje**

Rekomendacje wynikające z testów mogą obejmować:

- zalecane zmiany techniczne, takie jak rekonfiguracja sprzętu lub oprogramowania lub infrastruktury sieciowej,

- obszary zidentyfikowane do dalszej analizy (np. analiza logów serwera WWW, aby pomóc zidentyfikować podstawowe przyczyny problemów i/lub błędów),
- wymagane jest dodatkowe monitorowanie bram sieciowych, serwerów i sieci, aby można było uzyskać bardziej szczegółowe dane do pomiaru charakterystyk wydajności i trendów (np. degradacji).

# 5. Narzędzia — 90 min.

**Słowa kluczowe**
generator obciążenia, zarządzanie obciążeniem, narzędzie monitorujące, narzędzie do testów wydajnościowych

**Cele nauczania - narzędzia**

**5.1. Wsparcie narzędziowe**
PTFL-5.1.1. (K2) Kandydat rozumie, w jaki sposób narzędzia wspierają testowanie wydajnościowe.

**5.2. Przydatność narzędzi**
PTFL-5.2.1. (K4) Kandydat potrafi ocenić przydatność narzędzi do testowania wydajnościowego w danym scenariuszu projektowym.

## 5.1. Wsparcie narzędziowe

Narzędzia do testowania wydajnościowego obejmują następujące typy narzędzi wspierających testowanie wydajnościowe.

**Generatory obciążenia**

Generator, narzędzia IDE, edytor skryptów lub zestaw narzędzi, są w stanie tworzyć i wykonywać wiele instancji klienta, które symulują zachowanie użytkownika zgodnie ze zdefiniowanym profilem operacyjnym. Tworzenie wielu instancji w krótkich odstępach czasu spowoduje obciążenie testowanego systemu. Generator tworzy obciążenie, a także zbiera metryki do późniejszego raportowania.

Podczas wykonywania testów wydajnościowych celem generatora obciążenia jest naśladowanie świata rzeczywistego w możliwie największym stopniu. Często oznacza to, że potrzebne są żądania (and. *request)* użytkowników pochodzące z różnych lokalizacji, a nie tylko z lokalizacji testowej. Środowiska skonfigurowane z wieloma punktami obecności (ang. *points of presence*) będą dystrybuować się tam, skąd pochodzi obciążenie, tak, aby nie pochodziło ono w całości z jednej sieci. Zapewnia to realizm testu, chociaż czasami może wypaczyć wyniki, jeśli pośrednie przeskoki w sieci powodują opóźnienia.

**Konsola zarządzania obciążeniem**

Konsola zarządzania obciążeniem zapewnia sterowanie uruchamianiem i zatrzymywaniem generatora(-ów) obciążenia. Konsola zestawia również metryki z różnych transakcji, które są

zdefiniowane w instancjach obciążenia używanych przez generator. Konsola umożliwia przeglądanie raportów i wykresów z wykonania testów oraz wspiera analizę wyników.

**Narzędzie do monitorowania**

Narzędzia monitorujące działają równolegle z testowanym komponentem lub systemem i nadzorują, rejestrują i/lub analizują zachowanie komponentu lub systemu. Typowe monitorowane komponenty obejmują kolejki serwerów WWW, pamięć systemową i przestrzeń dyskową. Narzędzia do monitorowania mogą skutecznie wspierać analizę podstawowych przyczyn degradacji wydajności w testowanym systemie i mogą być również używane do monitorowania środowiska produkcyjnego po wydaniu produktu. Podczas wykonywania testów wydajnościowych można również używać monitoringu w samym generatorze obciążenia.

Modele licencji narzędzi do testów wydajnościowych obejmują tradycyjną licencję opartą na stanowiskach/lokacjach z pełnym prawem własności, model licencji typu *pay-as-you-go* w chmurze oraz licencje *open source*, z których można bezpłatnie korzystać w określonym środowisku lub w chmurze. Każdy model zakłada inną strukturę kosztów i może obejmować bieżące utrzymanie. Oczywiście w przypadku dowolnego, wybranego narzędzia, zrozumienie, jak działa to narzędzie (poprzez szkolenie i/lub samodzielną naukę), będzie wymagało czasu i budżetu.

## 5.2. Przydatność narzędzi

Wybierając narzędzie do testowania wydajnościowego należy wziąć pod uwagę następujące czynniki:

**Kompatybilność**

Na ogół narzędzie jest wybierane dla organizacji, a nie tylko dla projektu. Oznacza to uwzględnienie następujących czynników w organizacji:

- Protokoły: jak opisano w sekcji 4.2.1., protokoły są bardzo ważnym aspektem przy wyborze narzędzi do testów wydajnościowych. Zrozumienie, jakich protokołów używa system i które z nich będą testowane, dostarczy niezbędnych informacji do wyboru odpowiedniego narzędzia testowego.
- Interfejsy do komponentów zewnętrznych: interfejsy do komponentów oprogramowania lub innych narzędzi mogą być brane pod uwagę jako części wymagań dotyczących pełnej integracji, aby spełnić wymagania procesu lub inne wymagania dotyczące współdziałania (np. integracja z procesem CI).
- Platformy: niezbędna jest kompatybilność z platformami (i ich wersjami) w organizacji. Dotyczy to platform używanych do hostowania narzędzi oraz platform, z którymi narzędzia współdziałają w celu monitorowania i/lub generowania obciążenia.

### Skalowalność

Innym czynnikiem, który należy wziąć pod uwagę, jest całkowita liczba symulowanych, współbieżnych użytkowników, którą może obsłużyć narzędzie. Będzie to uwzględniać kilka czynników:

- maksymalna liczba wymaganych licencji,
- wymagania dotyczące konfiguracji stacji roboczej / serwera do generowania obciążenia,
- możliwość generowania obciążenia z wielu punktów obecności (np. serwery rozproszone).

### Zrozumiałość

Innym czynnikiem, który należy wziąć pod uwagę, jest poziom wiedzy technicznej potrzebny do korzystania z narzędzia. Jest to często pomijane i może prowadzić do sytuacji, w której niewykwalifikowani testerzy nieprawidłowo skonfigurują testy, co z kolei spowoduje otrzymanie niedokładnych wyników. W przypadku testów wymagających złożonych scenariuszy oraz wysokiego poziomu programowalności i dostosowywania, zespoły powinny upewnić się, że tester posiada niezbędne umiejętności, doświadczenie i przeszkolenie.

### Monitorowanie

Czy monitorowanie zapewniane przez narzędzie jest wystarczające? Czy w środowisku dostępne są inne narzędzia do monitorowania, które można wykorzystać do uzupełnienia monitorowania przez narzędzie? Czy monitorowanie może być skorelowane ze zdefiniowanymi transakcjami? Należy odpowiedzieć na wszystkie te pytania, aby określić, czy narzędzie zapewni wymagany monitoring przez projekt.

Gdy monitorowanie jest zapewnione przez oddzielny program/narzędzie/cały ekosystem, można go użyć do monitorowania środowiska produkcyjnego po wydaniu produktu.

# 6. Odniesienia

## 6.1. Standardy

- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE)

## 6.2. Dokumenty ISTQB®

- [ISTQB_UT_SYL] ISTQB® Foundation Level Usability Testing Syllabus, Version 2018
- [ISTQB_ALTA_SYL] ISTQB® Advanced Level Test Analyst Syllabus, Version 2012
- [ISTQB_ALTTA_SYL] ISTQB® Advanced Level Technical Test Analyst Syllabus, Version 2012
- [ISTQB_ALTM_SYL] ISTQB® Advanced Level Test Manager Syllabus, Version 2012
- [ISTQB_FL_SYL] ISTQB® Foundation Level (Core) Syllabus, Version 2018
- [ISTQB_FL_AT] ISTQB® Foundation Level Agile Tester Syllabus, Version 2014
- [ISTQB_GLOSSARY] ISTQB® Glossary of Terms used in Software Testing, http://glossary.istqb.org

## 6.3. Bibliografia

[Anderson01] Lorin W. Anderson, David R. Krathwohl (eds.) “A Taxonomy for Learning, Teaching and Assessing: A Revision of Bloom’s Taxonomy of Educational Objectives”, Allyn & Bacon, 2001, ISBN 978-0801319037

[Bath14] Graham Bath, Judy McKay, “The Software Test Engineer’s Handbook”, Rocky Nook, 2014, ISBN 978-1-933952-24-6

[Molyneaux09] Ian Molyneaux, “The Art of Application Performance Testing: From Strategy to Tools”, O’Reilly, 2009, ISBN: 9780596520663

[Microsoft07] Microsoft Corporation, “Performance Testing Guidance for Web Applications", Microsoft, 2007, ISBN: 9780735625709

# 7. Indeks