Metody testowania platformy KASKADA

Jerzy Proficz
Bartłomiej Daca
Tomasz Bieliński

12 września 2011

Streszczenie

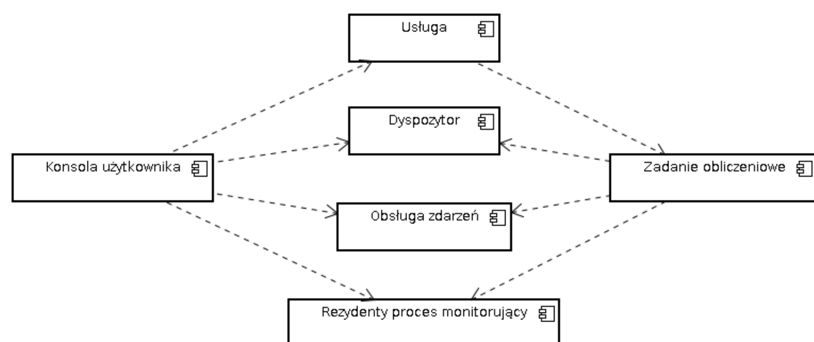
W rozdziale przedstawiono wykorzystywany iteracyjny i inkrementalny proces wytwarzania oprogramowania, ze szczególnym uwzględnieniem planowania, wykonywania i śledzenia testów oprogramowania platformy KASKADA. Zaprezentowano opis i przykłady wykorzystania testów jednostkowych, systemowych, wydajnościowych i wiarygodnościowych. Przedstawiono wybrane wyniki testów oraz ich wpływ na konstrukcję platformy.

Słowa kluczowe: Testy oprogramowania, testy jednostkowe, testy funkcjonalne, testy jakościowe, testy integracyjne, platforma KASKADA.

1 Charakterystyka metod testowania platformy KASKADA

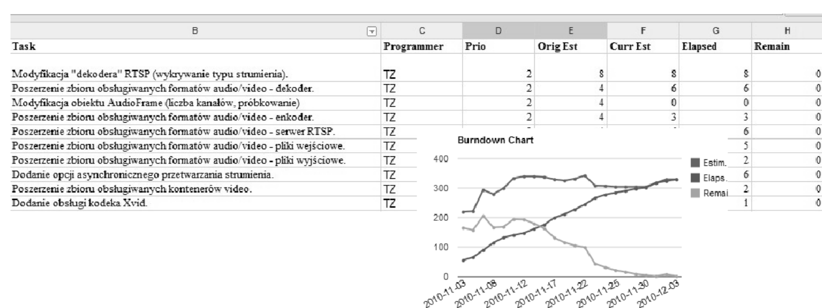
KASKADA jest złożoną platformą wspomagającą implementację, uruchamianie i wykonywanie algorytmów analizy i przetwarzania strumieni multimedialnych. Składa się z sześciu głównych komponentów programowych, osadzonych na pięciu skalowalnych serwerach, wykorzystuje wiele gotowych bibliotek i podsystemów: relacyjną bazę danych, system kolejek wiadomości, serwer serwletów, itp. Jej kod jest napisany w dwóch głównych językach programowania: Java i C++, liczących obecnie odpowiednio ok. 70 i 21 tyś. linii oraz 25 tyś. linii konfiguracji w XML'u.

Rysunek 1 przedstawia główne komponenty programowe platformy [5]. Konsola użytkownika jest wykorzystywana do zarządzania i konfiguracji platformy, usługi umożliwiają wyszukanie i wywoływanie zaprogramowanej funkcjonalności w postaci niezależnych komponentów, dyspozytor danych umożliwia odbiór, przesyłanie i wymianę strumieni multimedialnych, komponent obsługi zdarzeń jest odpowiedzialny za wspomaganie komunikacji wewnątrz platformy jak również ze światem zewnętrznym, zadania obliczeniowe wykonują operacje analizy i przetwarzania danych w tym strumieni multimedialnych, a ich wykonywanie jest kontrolowane przez rezydentny proces monitorujący. Każdy z powyższych komponentów przedstawia inne wymagania dla testów. Komponenty współpracujące bezpośrednio z użytkownikiem, np. konsola użytkownika wymagają innego podejścia do testowania niż komponenty obsługujące dalsze warstwy przetwarzania, np. zadania obliczeniowe, czy monitorujące.



Rysunek 1: Główne komponenty platformy KASKADA

W pracach nad platformą KASKADA bierze udział dziesięcioosobowy zespół programistów i testerów. Członkowie zespołu współpracują ze sobą wykorzystując zalecenia zawarte w procesie RUP jak również elementy metodologii zwinnych [1] (ang. agile). Prace są zorganizowane w iteracje, w każdej z nich dodawana jest przyrostowo nowa funkcjonalność. Składają się one z etapu wytworzenia przyrostu oprogramowania, oraz z etapu jego testowania. W etapie drugim przeprowadzane są testy oraz wprowadzane poprawki eliminujące wykryte błędy (ang. bug fixing). Każda iteracja posiada swój plan działania – listę zadań związanych z implementowanymi funkcjonalnościami wraz z przypisanymi programistami/testerami. Rysunek 2 przedstawia przykład takiego planu dla etapu tworzenia oprogramowania wraz z wykresem obrazującym postęp prac. Oprócz planów, w projekcie wytwarzana jest typowa dokumentacja: specyfikacja wymagań, architektura systemu, projekty szczegółowe zgodnie z dobrymi zasadami inżynierii oprogramowania.



B	C	D	E	F	G	H
Task	Programmer	Prio	Orig Est	Curr Est	Elapsed	Remain
Modyfikacja "dekodera" RTSP (wykrywanie typu strumienia).	TZ	2	8	8	8	0
Poszerzenie zbioru obsługiwanych formatów audio/video - dekoder.	TZ	2	4	6	6	0
Modyfikacja obiektu AudioFrame (liczba kanałów, próbkowanie)	TZ	2	4	0	0	0
Poszerzenie zbioru obsługiwanych formatów audio/video - enkoder.	TZ	2	4	3	3	0
Poszerzenie zbioru obsługiwanych formatów audio/video - serwer RTSP.	TZ				6	0
Poszerzenie zbioru obsługiwanych formatów audio/video - pliki wejściowe.	TZ				5	0
Poszerzenie zbioru obsługiwanych formatów audio/video - pliki wyjściowe.	TZ				2	0
Dodanie opcji asynchronicznego przetwarzania strumienia.	TZ				6	0
Poszerzenie zbioru obsługiwanych kontenerów video.	TZ				2	0
Dodanie obsługi kodeka Xvid.	TZ				1	0

Rysunek 2: Fragment planu zadań dla przykładowej iteracji

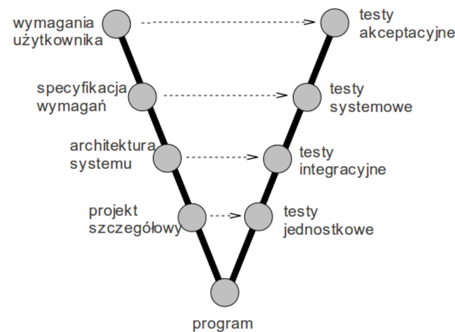
Testowanie jest procesem wykonywania programu w celu znalezienia w nim błędów. Testy są opisywane przez przypadki testowe – scenariusze wykonania

danej funkcjonalności, dla konkretnych danych i parametrów wejściowych, przy założonym wyniku działania. Dobry przypadek testowy to taki, przy wykonaniu którego prawdopodobieństwo wykrycia błędu jest wysokie. Dobrze zaprojektowane testy umożliwiają systematyczne wykrywanie różnych klas błędów, minimalizując czas i wysiłek ich wykonania [8]. Istnieją dwie główne strategie testowania: *white box* i *black box*. Pierwsza z nich zakłada znajomość wewnętrznej struktury testowanego oprogramowania, a przypadki testowe powstają na podstawie kodu źródłowego lub jego projektu. Natomiast druga strategia opiera się na traktowaniu testowanego oprogramowania jako zbioru funkcji, które muszą wykonać stawiane im zadania przy założonych wymaganiach jakościowych. Dla tego podejścia przypadki testowe powstają na podstawie wymagań funkcjonalnych i jakościowych, bez informacji na temat wewnętrznej budowy oprogramowania [6].

Platforma KASKADA, tak jak każdy złożony program wykorzystujący paradygmaty przetwarzania równoległego i rozproszonego, jest narażony na szczególne problemy związane ze współbieżnym wykonywaniem poszczególnych jego elementów: wątków, procesów czy usług. Duże nasilenie komunikacji, chwilowe przeciążenie sieci komputerowej czy zmienne opóźnienia spowodowane obliczeniami może spowodować, że jednoczesny dostęp do pamięci czy odbiór poszczególnych komunikatów może nastąpić w różnej kolejności, powodując wystąpienie zjawiska wyścigów oraz spowodowanych przez nie błędów zależnych czasowo. Z tego powodu konieczne było wzbogacenie zestawu testów o przypadki testowe umożliwiające wykrycie takich problemów [4].

Rysunek 3 przedstawia model „V” przeprowadzania testów, w którym poszczególne produkty (ang. *artifacts*) procesu wytwarzania oprogramowania służą do projektowania przypadków testowych dla różnych rodzajów testów [6]. Projekt szczegółowy jest wykorzystany do wykonywania testów jednostkowych, które zgodnie z obecnie popularną metodologią zwinną [1], powinny być przygotowywane jeszcze przed napisaniem właściwego kodu. Testy integracyjne, wykonywane na podstawie dokumentu architektury systemu, umożliwiają weryfikację protokołów współpracy komponentu systemu oraz bibliotek zewnętrznych. Na kolejnym poziomie wykonywane są testy systemowe. Projektuje się je na podstawie specyfikacji wymagań: wymagania funkcjonalne są wykorzystywane przy testach funkcjonalnych, natomiast wymagania niefunkcjonalne umożliwiają przygotowanie testów jakościowych. Na końcu przeprowadzane są testy akceptacyjne bezpośrednio przez użytkownika końcowego.

W przypadku platformy KASKADA, wykonywane są testy na wszystkich opisanych poziomach. W następnym podrozdziale opisane zostały metody wykonywania testów jednostkowych oraz integracyjnych wspomaganych przez automatyczne narzędzia JUnit i Boost Test. Testy funkcjonalne przeprowadzane przez testerów na podstawie zestawu przygotowanych przypadków testowych, zostały one opisane w kolejnym podrozdziale. Następnie zaprezentowano sposoby wykonywania testów jakościowych dla wybranych charakterystyk platformy. Na końcu rozdziału zawarto podsumowanie dotyczące dojrzałości platformy KASKADA.



Rysunek 3: Model „V": wykorzystanie produktów procesu wytwarzania oprogramowania do testów

2 Testy jednostkowe i integracyjne

Testy jednostkowe i integracyjne stanowią pierwszy z etapów testowania, któremu poddawana jest platforma KASKADA. Naturalnym jest więc, że to właśnie na tym poziomie powinno być wykryte najwięcej błędów. Wszelkie niewychwycone błędy stają się na dalszych etapach o wiele trudniejsze do wykrycia, a co za tym idzie, bardziej kosztowne do wyeliminowania. Dlatego tak ważne jest, by testy te były zaprojektowane w sposób, który pokrywałby jak najwięcej możliwych ścieżek od wierzchołka startowego do wierzchołka końcowego grafu przepływu wykonywania aplikacji. [9]

Testy te są testami typu *white box*, a więc ich tworzenie oparte jest na znajomości: wewnętrznej architektury systemu w przypadku testów integracyjnych, bądź projektu szczegółowego w przypadku testów jednostkowych. To właśnie dzięki zastosowaniu tej techniki możliwa jest maksymalizacja liczby ścieżek pokrytych testami.

W przypadku platformy KASKADA testy jednostkowe i integracyjne są w pełni zautomatyzowane. Kluczową rolę w procesie wytwarzania platformy KASKADA spełnia serwer budujący (ang. *build server*) hudson. Jeżeli którykolwiek z testów nie zakończy się sukcesem, to do odpowiedzialnych za komponent deweloperów jest wysyłana informacja drogą mailową zawierająca szczegóły błędu, takie jak nazwa testu i warunek, który nie został spełniony. Rysunek 4 pokazuje przykładowe wysłanie wiadomości do programisty w reakcji na błąd w jednym z testów.

2.1 Testy jednostkowe

Podczas tworzenia platformy KASKADA deweloperzy wykonują krótkie cykle procesu wytwarzania zgodne ze schematem ukazanym na Rysunku 5. Rozpoczynają od implementacji testu, który sprawdza poprawność powstającej funkcjonalności. Ważnym jest, by początkowo test ten kończył się porażką. Jeżeli jest

```
13/19 Test #13: amq_messenger_test .........................   Passed    2.73 sec
      Start 14: audio_frame_test
14/19 Test #14: audio_frame_test ...........................   Passed    0.03 sec
      Start 15: monitor_util_test
15/19 Test #15: monitor_util_test ..........................   Passed    0.20 sec
      Start 16: monitor_xml_test
16/19 Test #16: monitor_xml_test ...........................   Passed    0.10 sec
      Start 17: node_monitor_test
17/19 Test #17: node_monitor_test ..........................   Passed   16.25 sec
      Start 18: task_monitor_xml_test
18/19 Test #18: task_monitor_xml_test ......................   Passed    0.11 sec
      Start 19: task_start_info_object_serialization_test
19/19 Test #19: task_start_info_object_serialization_test ...   Passed    0.44 sec

95% tests passed, 1 tests failed out of 19

Total Test time (real) =  90.41 sec

The following tests FAILED:
          8 - socket_binary_inout_stream_handler_test (SEGFAULT)
Errors while running CTest
make: *** [test] Error 8
Sending e-mails to: bdaca@task.gda.pl
Finished: UNSTABLE
```

Rysunek 4: Fragment wyników testów przeprowadzanych na serwerze budującym hudson

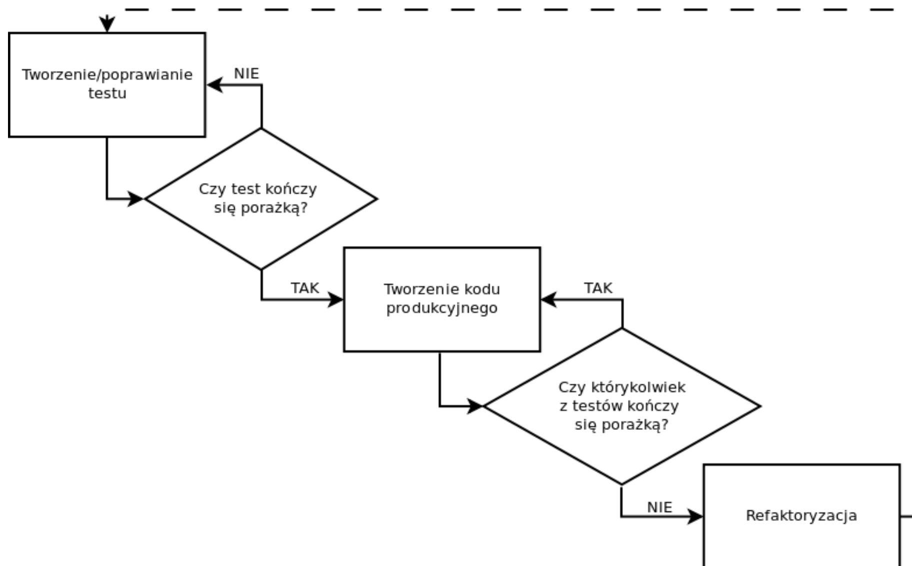
inaczej oznacza to, że albo funkcjonalność już została wcześniej zaimplementowana, albo, co bardziej prawdopodobne, test został błędnie skonstruowany. Kolejnym krokiem jest implementacja klasy bądź metody odpowiadającej za dodawaną funkcjonalność. Ta kończy się sukcesem w momencie, gdy napisany w kroku pierwszym test daje wynik pozytywny. Proces kończy się refaktoryzacją kodu w celu poprawy jego jakości, przy której deweloper powinien posiłkować się testami, by upewnić się, że podczas procesu oczyszczania kodu nie doszło do uszkodzenia funkcjonalności. Wytwarzanie oprogramowania w taki sposób nazywane jest wytwarzaniem sterowanym testami (ang. *TDD – test-driven development*) [2].

Testy jednostkowe, zgodnie ze swoją nazwą, powinny dotyczyć tylko jednego elementu. Są one grupowane w zbiory testów, które dotyczą danej klasy. Jeżeli do osiągnięcia celu konieczna jest komunikacja z innymi komponentami systemu, tworzy się atrapy (ang. *mocks*). Do tego celu w komponentach tworzonych w języku Java używany jest pakiet jMock. W przypadku języka C++ do tworzenia atrap nie jest używany żaden specjalny framework. Są za to tworzone klasy, które symulują pewne części systemu. Ich nazwy zawsze zaczynają się od przedrostka „Dummy", np. *DummyMessenger*.

Testy jednostkowe są w pełni zautomatyzowane i wykonywane przez serwer budujący, który co pięć minut sprawdza, czy do repozytorium wpłynęła nowa wersja kodu i jeżeli wykryje taką sytuację, rozpoczyna budowę i testy zmodyfikowanych komponentów.

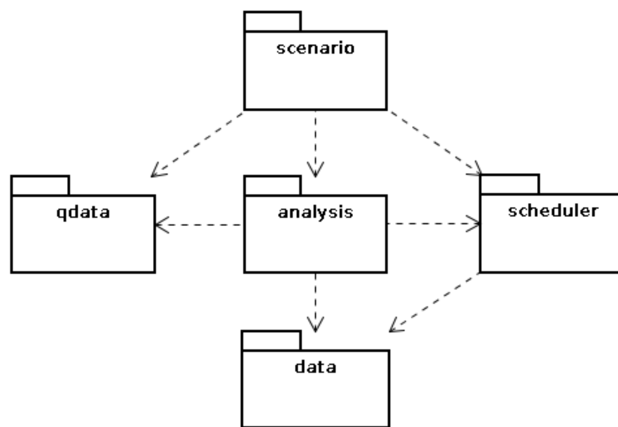
2.2 Testy integracyjne

Po wykonaniu testów jednostkowych i zintegrowaniu przetestowanych klas z rzeczywistymi podsystemami wykonywane są testy integracyjne. Pozwalają one wykryć błędy w komunikacji między komponentami. Podobnie jak testy jednostkowe, testy integracyjne wykonywane są na serwerze budującym. Odbywa się to codziennie o godzinie 2:00. Testowane są interakcje z bazą danych, jak również wykorzystanie WSDL i UDDI. Testy bazodanowe są oparte na silniku



Rysunek 5: Schemat blokowy wytwarzania oprogramowania sterowanego testami (TDD)

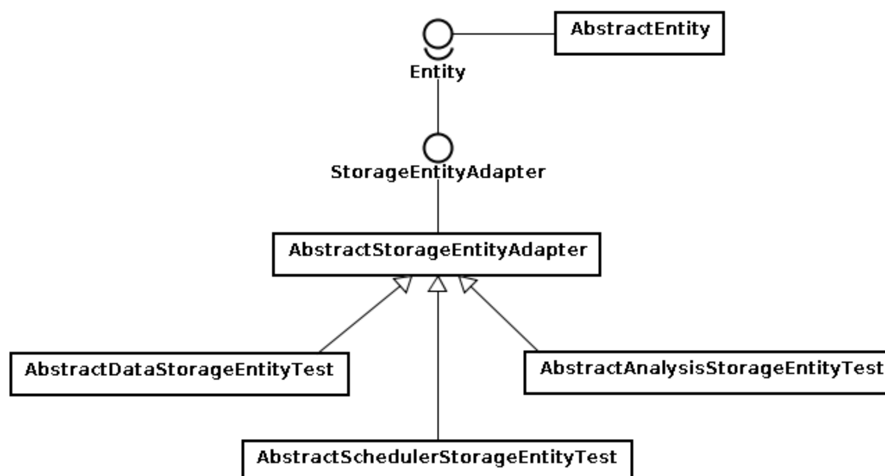
hsqldb, przechowującym swoje dane i strukturę w pamięci procesu. Dzięki temu możliwe jest tworzenie jej za każdym razem na nowo, co ułatwia utrzymanie takich samych warunków początkowych dla każdego testu.



Rysunek 6: Architektura komponentu mayday.uc

Architektura testów konsoli użytkownika powiązana jest z architekturą kom-

ponentu mayday.uc (patrz Rysunek 6). Poszczególne pakiety są odpowiedzialne za różne elementy konsoli użytkownika (np. pakiet *qdata* odpowiada za parametry jakościowe usług). Każdy z elementów, o którym dane są trzymane w bazie danych, a na które wpływ ma użytkownik, dziedziczy po interfejsie *Entity*. Do testowania zaś używane są klasy dziedziczące po interfejsie *StorageEntityAdapter*. Każdy z pakietów komponentu ma swoją implementację tego interfejsu. Opisane zależności przedstawione są na Rysunku 7. Dzięki takiemu rozwiązaniu możliwe jest zmniejszenie czasu i zasobów zużywanych podczas procesu testowania, gdyż do wykonania testów na poszczególnych pakietach możliwe jest odtworzenie tylko tego fragmentu bazy danych, z którym zachodzi interakcja.



Rysunek 7: Diagram klas reprezentujący architekturę testowania komponentu mayday.uc

3 Testy funkcjonalne

Testy funkcjonalne platformy KASKADA są zrealizowane w oparciu o paradygmat *black box*, który zakłada brak informacji na temat budowy wewnętrznej oprogramowania. Są one częścią testów Systemowych z modelu "V" (Rysunek 3). Pozwalają na zweryfikowanie czy testowane oprogramowanie jest zgodne z specyfikacją wymagań. Testy funkcjonalne zostały zaprojektowane na podstawie specyfikacji przypadków użycia. Z tego powodu tester wykonujący testy funkcjonalne zazwyczaj korzysta z dwóch dokumentów - opisującego przypadki użycia i drugiego opisującego przypadki testowe.

Dobrym przykładem przypadku użycia może być "Zarządzanie węzłami klastra" przedstawione na Rysunku 8. W dokumencie opisującym ten przypadek został określony scenariusz jego wykonania, wraz z alternatywnymi przebie-

Rysunek 8: Przykładowe przypadki użycia platformy KASKADA

gami. Ponadto wyspecyfikowane są warunki początkowe. Fragment takiego dokumentu został uwidoczniony poniżej:

Warunki wstępne:
1.Administrator jest uwierzytelniony i autoryzowany

Scenariusz podstawowy:
1.Administrator wybiera opcję dodania danych nowego węzła klastra.
2.System przedstawia pusty formularz z opisem węzła.
3.Administrator uzupełnia formularz.
4.System sprawdza poprawność wpisanych danych.
5.System dodaje wprowadzone dane węzła.
6.System przedstawia listę węzłów, dla których przechowuje dane.
7.Administrator wybiera węzeł dla którego będzie modyfikować dane.
...

Scenariusz alternatywny, niepoprawne dane przy dodawaniu danych węzła:
kroki 1-4
5a.System informuje administratora o niepoprawnych danych wprowadzonych w formularzu.

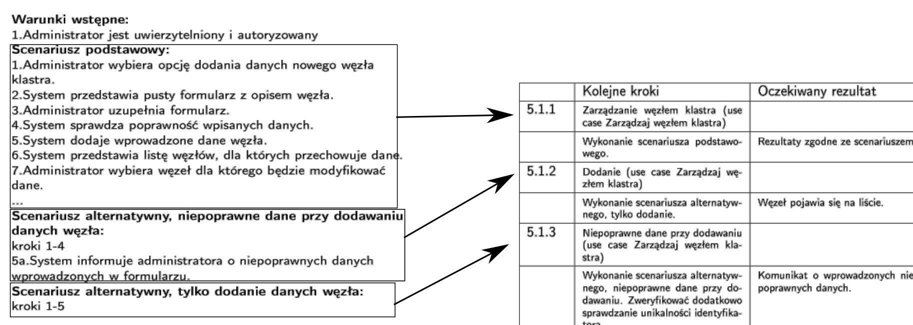
Scenariusz alternatywny, tylko dodanie danych węzła:
kroki 1-5

W Tabeli 1 przedstawiono kilka przykładowych przypadków testowych zaprojektowanych dla powyższego przypadku użycia. Opisane są w niej czynności jakie ma wykonać tester w testowanej platformie. Zazwyczaj w czasie projektowania testów funkcjonalnych tworzy się jeden przypadek testowy na każdy scenariusz wymieniony w specyfikacji przypadków użycia. Przykładem może tu być przyporządkowanie wykonane dla przypadku 5.1 przedstawione na Rysunku 9. Opisy kolejnych kroków w Tabeli 1 odnoszą się, zgodnie z przy-

Tablica 1: Wybrane przypadki testowe dla "Zarządzanie węzłami klastra".

	Kolejne kroki	Oczekiwany rezultat
5.1.1	Zarządzanie węzłem klastra (use case Zarządzaj węzłem klastra)	
	Wykonanie scenariusza podstawowego.	Rezultaty zgodne ze scenariuszem.
5.1.2	Dodanie (use case Zarządzaj węzłem klastra)	
	Wykonanie scenariusza alternatywnego, tylko dodanie.	Węzeł pojawia się na liście.
5.1.3	Niepoprawne dane przy dodawaniu (use case Zarządzaj węzłem klastra)	
	Wykonanie scenariusza alternatywnego, niepoprawne dane przy dodawaniu. Zweryfikować dodatkowo sprawdzanie unikalności identyfikatora.	Komunikat o wprowadzonych niepoprawnych danych.

porządkowaniem, do scenariuszy przypadków użycia, zaś w kolumnie obok są przedstawione oczekiwane rezultaty testów.



Rysunek 9: Przyporządkowanie scenariuszy przypadku użycia 5.1 „Zarządzanie węzłami klastra“ do przypadków testowych

Ze względu na ograniczony skład zespołu testy funkcjonalne są wykonywane przez osoby piszące oprogramowanie. Zadania te są tak dobierane, aby testy nie były wykonywane przez autora odpowiadającego im kodu po to, aby można było spełnić paradygmat *black box*. Osoba wykonująca testy funkcjonalne przyjmuje rolę użytkownika, dewelopera lub administratora w zależności od warunków początkowych dla danego przypadku testowego. Jest to bezpośrednie powiązanie z diagramem przypadków użycia. Zadanie testera polega na wykonaniu krok po kroku czynności odpowiadających danemu przypadkowi użycia i sprawdzeniu czy otrzymane rezultaty pokrywają się z oczekiwanymi. Np. dla przypadku 5.1.3, tester ma za zadanie sprawdzić jak system reaguje

na wpisanie niepoprawnych danych w formularzu dodawania węzła. Po wpisaniu błędnych danych osoba testująca musi stwierdzić, czy platforma zwróciła komunikat o błędzie. Jeśli nie oznacza to wykrycie błędu.

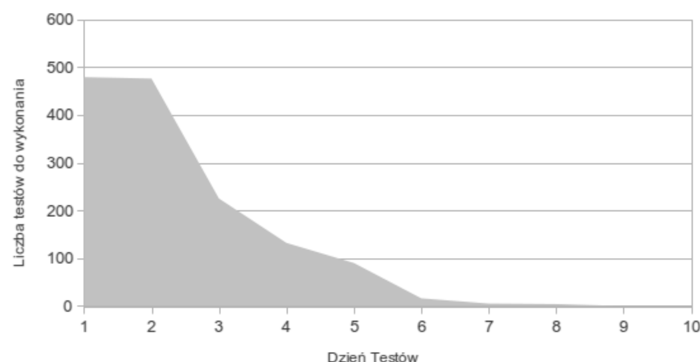
2.8.1.3	RK	d	
2.8.1.4	RK	d	Niemożliwe podanie błędnych danych
2.8.1.5	RK	d	
2.8.1.6	RK	d	
2.8.1.7	RK	d	
2.8.1.8	RK	d	
2.9.1	RK	d	Brak PID dla zadań
2.9.2	RK	d	
2.9.3	RK	d	
2.9.4	RK	d	
2.9.9	RK	d	
2.9.10	RK	d	
3.1.1	TZ	d	
3.1.2	TZ	d	
3.1.3	TZ	d	
3.1.4	TZ	d	
3.1.5	TZ		#222
3.1.6	TZ	d	

Rysunek 10: Fragment raportu z wykonania testów funkcjonalnych dla platformy KASKADA

W czasie przeprowadzania testów funkcjonalnych platformy KASKADA stosowane są dwa narzędzia do komunikacji w zespole. Pierwszym z nich jest raport z wykonania testów funkcjonalnych (Rysunek 10) w postaci współdzielonego dokumentu. Składa się on z czterech kolumn, które dla każdego przypadku testowego zawierają następujące informacje: nr przypadku testowego, inicjały testera wykonującego przypadek testowy, informację o tym czy przypadek testowy został wykonany pomyślnie oraz komentarz testera. Drugim narzędziem jest Trac [15], czyli system zgłaszania błędów. Tester który odkrył błąd, opisuje go w specjalnym formularzu narzędzia Trac. Umieszcza tam informacje o okolicznościach jego pojawienia się, oraz specyfikuje krok po kroku czynności jakie należy wykonać aby doprowadzić do ponownego ujawnienia się błędu. Następnie tak utworzoną informację (ang. *ticket*) przypisuje do osoby odpowiedzialnej za dany fragment platformy, a także umieszcza nr błędu w raporcie w polu z komentarzem (Rysunek 10, przypadek 3.1.5). Deweloper, który otrzymał raport o błędzie, wykonuje odpowiednie poprawki w kodzie. Następnie odsyła wiadomość do testera. Osoba ta przeprowadza test, który wykaże, czy błąd został usunięty. Jeśli testy się powiodą to błąd uznawany jest za wyeliminowany. Jeśli jednak testy się nie powiodą, to do dewelopera odpowiedzialnego za tą naprawę jest wysyłane powiadomienie o konieczności dokonania poprawek.

W czasie przeprowadzania testów funkcjonalnych testerzy oznaczają w raporcie (Rysunek 10) w kolumnie trzeciej, które testy zostały już wykonane, poprzez wpisanie litery „d" (skrót od ang. *done*). Pozawala to na ułożenie wykresu postępu wykonania testów (Rysunek 11).

W procesie testowania platformy KASKADA stosowane są dwa środowiska. Pierwszym jest środowisko deweloperskie (Dev), w którym są wykonywane testy funkcjonalne przez testerów z zespołów deweloperskiego. Drugim jest środowi-



Rysunek 11: Przykładowy wykres postępu wykonania testów funkcjonalnych dla platformy KASKADA

sko do testów akceptacyjnych (Test), które są wykonywane przez użytkowników i deweloperów z innych zespołów projektu MAYDAY EURO 2012. Testy funkcjonalne platformy są przeprowadzane w drugiej części każdej iteracji. Pozwalają formalnie potwierdzić poprawność funkcjonalną platformy.

4 Testy jakościowe

Platon definiuje jakość jako stopień doskonałości. Jak zauważono w [6], w stosunku do produktów informatycznych wielu niesłusznie pojmuje jakość jako spełnienie wymagań funkcjonalnych produktu. Jest to jednak spojrzenie niezwykle wąskie, ograniczające się do definicji jakości związanej z produkcją, pomijana jest definicja jakości związana m. in. z produktem czy użytkownikiem [3]. Nie wszystkie aspekty jakości są łatwo mierzalne, bowiem różne parametry jakościowe, głównie te postrzegane przez użytkownika, są subiektywne.

Testy jakościowe skupiają się na ocenie niezawodności, wiarygodności, wydajności i skalowalności platformy KASKADA. Przeprowadzane są na kolejnych publikowanych wersjach platformy. Ponadto na bieżąco ocenie poddawane są parametry subiektywne jakości pracy z platformą – użytkownicy pracujący na kolejnych wersjach zgłaszają swoje uwagi co do ergonomii pracy czy estetyki wykonania. Uwagi te są następnie poddawane ocenie przez zespół projektowy i, jeżeli została podjęta decyzja o realizacji, wprowadzane w kolejnej wersji platformy.

4.1 Zasady Testowania Jakościowego

Według definicji zamieszczonej w słowniku języka polskiego [13], obiekt niezawodny to „taki, na którym zawsze można polegać”. W platformie KASKADA

niezawodność uważana jest za ważną cechę, opisaną w specyfikacji systemu. Odnosi się ona głównie do niezawodności sprzętu, na którym pracuje i odpowiedniego reagowania na wszelkie niestabilności w działaniu warstwy sprzętowej. Brane są pod uwagę dwa scenariusze: gdy węzeł, na którym wykonywane były usługi zostaje odłączony od sieci bądź całkowicie wyłączony. W obu przypadkach wszelkie pozostałe zadania w ramach usług, których część była wykonywana na ulegającym awarii węźle muszą zostać zakończone odpowiednim komunikatem o błędzie, a dane o nich usunięte z bazy danych. Ponadto, w przypadku gdy węzeł ponownie zostaje wpięty do topologii sieci, z której wcześniej został odłączony, powinny na nim zostać zatrzymane wszelkie działające wcześniej zadania.

Testowanie poprawności przedstawionego powyżej zachowania odbywa się przed każdym opublikowaniem nowej wersji platformy KASKADA. Jest wykonywane na rzeczywistym sprzęcie, odłączenie od sieci symulowane jest za pomocą wyłączenia interfejsu sieciowego, a wyłączenie węzła poprzez restart maszyny.

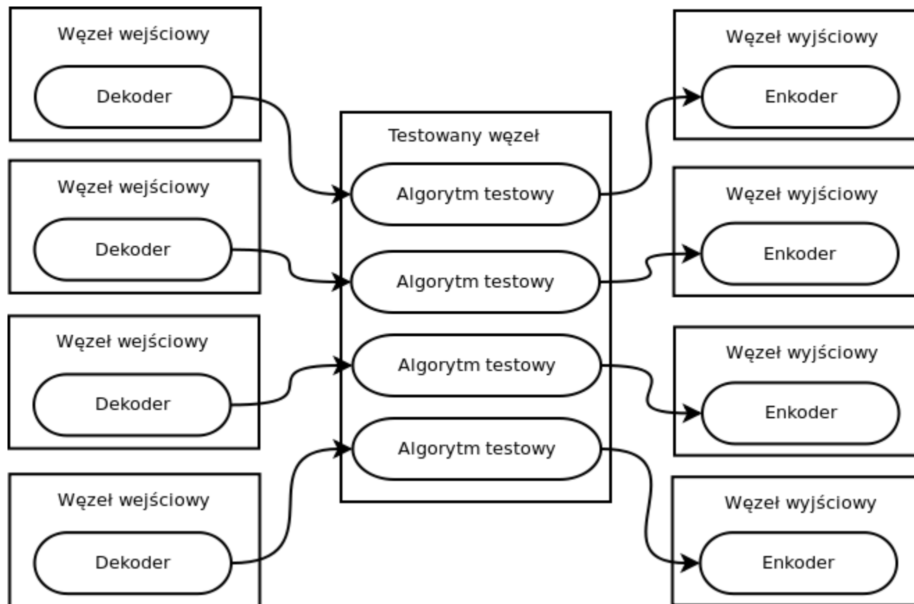
4.2 Charakterystyka testów wiarygodności, wydajności i skalowalności

Podczas testowania pozostałych trzech parametrów wykonywane są eksperymenty testowe polegające na uruchamianiu różnej – w zależności od testowanej cechy – liczby zadań i obserwacji:

- obciążenia procesora – testowanie wydajności i skalowalności;
- zużycia pamięci – testowanie wydajności i skalowalności;
- ruchu przychodzącego – testowanie wydajności i skalowalności;
- ruchu wychodzącego – testowanie wydajności i skalowalności;
- liczby ramek w pliku wyjściowym – testowanie wiarygodności i skalowalności.

Architektura każdego przypadku testowego opiera się na rozłożeniu zadań pomiędzy węzły wejściowe, węzeł testowy oraz węzły wyjściowe (Rysunek 12), pełniące następujące funkcje:

- węzły wejściowe – dokonujące odczytu strumienia wejściowego, dekodujące go i następnie przesyłające do węzła testowego;
- węzeł testowy – węzeł, na którym dokonywane były pomiary; na nim uruchamiane były algorytmy odpowiednie dla danego eksperymentu;
- węzły wyjściowe – aby umożliwić zliczanie ramek, zachodzi konieczność kodowania i zapisu strumieni wyjściowych na dysk, co realizowane jest właśnie przez te węzły.



Rysunek 12: Schemat infrastruktury sprzętowej dla testów jakościowych

Liczba węzłów wejściowych i wyjściowych jest równa liczbie uruchamianych zadań w danym teście, aby nie zniekształcić wyników testów. Do eksperymentów używane są strumienie testowe, zarówno audio i wideo, o różnych parametrach. Przykładowy taki zbiór przedstawiony jest w Tabeli 2.

Tablica 2: Przykładowe parametry strumieni do używanych podczas testów.

wideo			
Wymiary klatki [px]	Częstotliwość [fps]	Czas trwania [mm:ss]	Liczba klatek
704x576	21,7	05:00	6506
1920x1080	30,9	05:00	9262
audio			
Rozmiar pakietu [B]	Częstotliwość	Czas trwania [mm:ss]	Liczba klatek
9216	38,3	05:00	11485

4.3 Wiarygodność

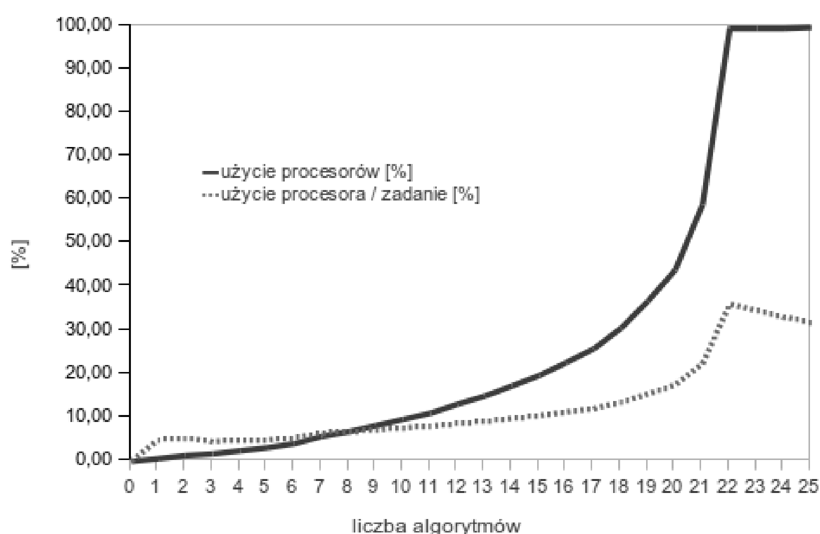
Wiarygodność systemu to cecha określająca stopień zaufania do usług oferowanych przez ten system [12]. W platformie KASKADA, której zadaniem jest kontekstowa analiza danych strumieniowych, jej ocena jest wyznaczana na pod-

stawie liczby utraconych pakietów danych. Wymaganie, jakie zostało nałożone na platformę, to utrata maksymalnie 1% klatek wideo lub próbek audio.

Testowanie wiarygodności polega na sprawdzeniu liczby klatek strumienia testowego i porównanie jej z liczbą klatek strumienia wyjściowego, będącego rezultatem przetwarzania przez algorytm testowy. Procent utraconych klatek może być różny w zależności od użytego do testowania algorytmu. Jeżeli algorytm wymaga dużej mocy obliczeniowej, możliwa jest sytuacja, gdy czas przetwarzania będzie większy niż okres, z jakim do zadania dochodzą kolejne elementy strumienia. Nie bez znaczenia jest również strumień danych – im elementy strumienia mają większy rozmiar, tym wymagany czas na ich transfer jest dłuższy. Dlatego wykonywanych jest wiele kombinacji – testowane są algorytmy o różnym stopniu złożoności i strumienie o różnych rozmiarach.

4.4 Skalowalność

Skalowalność to cecha systemu umożliwiająca zwiększenie jego wydajności na skutek rozbudowy jego komponentów [7]. W platformie KASKADA testy skalowalności polegają na stopniowym zwiększaniu liczby jednocześnie uruchamianych algorytmów i obserwacji obciążenia sieci, pamięci i procesora (Rysunek 13). Liczba ta jest zwiększana tak długo, jak długo platforma działa poprawnie, a ocena wiarygodności mieści się w założonych granicach (poniżej 1% utraty danych).



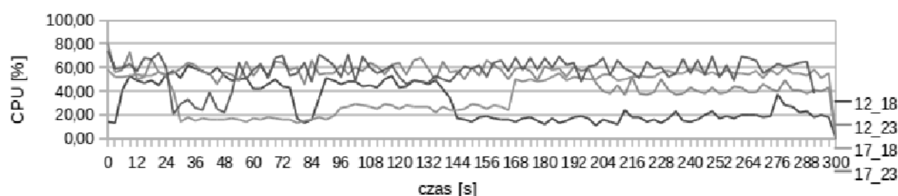
Rysunek 13: Wykres obciążenia procesorów w zależności od ilość procesów *kaskada_relayer* uruchomionych na węźle obliczeniowym

Pomiar obciążenia procesora oraz zużycia pamięci dokonywany jest z uży-

ciem polecenia systemu Linux *top* [14]. Przez cały czas trwania testu co sekundę zbierane są próbki, z których każda zawiera informację o średnim użyciu procesora/pamięci od ostatniego dokonanego pomiaru. Następnie wyliczana była średnia z tych próbek. Ponieważ w sieci typu *InfiniBand* [11] mogą być wykorzystywane metody transportu przezroczyste dla procesora (np. RDMA), nie jest możliwe korzystanie ze standardowych liczników obciążenia sieci (np. poleceniem *ifconfig*) Dlatego pomiar wykonywany jest z wykorzystaniem liczników znajdujących się bezpośrednio na kartach sieciowych. Zliczają one czwórki bajtów, które zostały wysłane z bądź dostarczone do interfejsu. Tak otrzymane dane z eksperymentów o różnej liczbie uruchomionych zadań wykonujących założone algorytmy można łatwo porównać i umieścić na wykresie, co pozwala wyznaczyć trend wzrostu poszczególnych mierzonych cech.

4.5 Wydajność

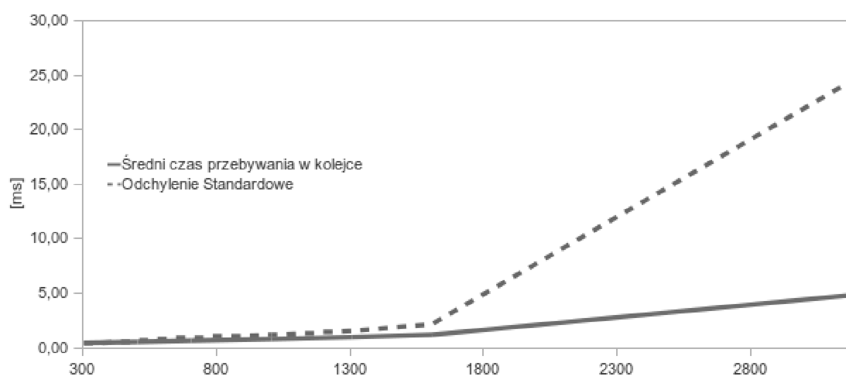
Testy wydajności skupiają się na badaniu trzech aspektów: wydajności pojedynczego zadania, klastra, na którym pracuje platforma oraz serwera ActiveMQ [10] używanego do komunikacji pomiędzy zadaniami (patrz architektura platformy [5]). Testy wydajności zadań, podobnie jak skalowalności, uruchamiane są na pojedynczym węźle, ale tylko z jednym zadaniem testowym. Metoda gromadzenia danych pomiarowych pozostaje taka sama jak w przypadku testów skalowalności. Otrzymane próbki wykorzystywane są do wyliczenia wartości średnich użycia procesora, pamięci i sieci, a także do wykonania wykresów, ukazujących jak kształtowała się dana cecha w czasie trwania testu – Rysunek 14. Pozwala to odnaleźć specyficzne błędy jakościowe, takie jak wycieki pamięci.



Rysunek 14: Wykres wynikowy jednego z testów wydajności

Do testów całościowych klastra użyte zostały usługi o szacunkowym obciążeniu 900 i 6100 punktów procentowych mocy pojedynczego rdzenia. Sumaryczna wydajność używanego klastra – 231 węzłów posiadających po 8 rdzeni każdy – wynosi 184800 punktów procentowych. Testy, w zależności od eksperymentu, polegają na próbie uruchomienia teoretycznie maksymalnej liczby usług rodzaju pierwszego, drugiego oraz obu jednocześnie, zarówno z, jak i bez strumienia wyjściowego. Testy te umożliwiają wykrycie wąskich gardeł całego systemu.

Testowanie wydajności serwera ActiveMQ polega na uruchamianiu dużej (co najmniej 200) liczby zadań wysyłających do niego różne rodzaje wiadomości. Sposób wysyłania wiadomości zależy od eksperymentu i może być: okresowe



Rysunek 15: Wykres średniego czasu przebywania wiadomości w kolejce i odchylenia standardowego tego czasu od ilości klientów dla symulacji zadań wysyłających wiadomości z częstotliwością 1 wiadomość/10 s do klastra ActiveMQ złożonego z dwóch serwerów poprzez sieć InfiniBand.

z częstotliwością 1/5 s, okresowe z częstotliwością 1/10s (Rysunkek 15), oraz wybuchowe (ang. Burst) z częstotliwością 1/s przez 10s, a następnie 20s oczekiwania.

Ostatnią kategorią testów wykonywanych na platformie są testy akceptacyjne. Ich przygotowywaniem zajmują się bezpośrednio użytkownicy platformy: członkowie zespołów wykonujących aplikacje pilotażowe w projekcie MAYDAY EURO 2012. Ich ocena jest końcowym elementem zamykającym każdą iterację, a akceptacja bardzo ważnym czynnikiem wskazującym na jakość wykonania całej platformy KASKADA. Powyżej przedstawione metody powstały na podstawie dobrze opisanych procesów wytwarzania oprogramowania (RUP, metodologie zwinne [1]), jak również zostały wzbogacone o elementy wynikające z doświadczeń całego zespołu. Oczywistym jest, że powyższy proces wymaga kolejnych ulepszeń i rozwoju w celu podwyższenia jakości wytwarzanych komponentów. Dobrym rozwiązaniem byłoby wydzielenie osobnego zespołu wykonującego testy systemowe, jak również wprowadzenie bardziej formalnych metod zachowania jakości procesu wytwarzania oprogramowania.

Literatura

[1] Amber S. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*, Wiley, 2002

[2] Beck K. *Test-Driven Development by Example*, Addison Wesley, 2003

[3] Garvin G. A. *What does product quality really mean*, Sloan Management Review, Cambridge, 1984

[4] Krawczyk H., Proficz J. *Casuality checking procedures for distributed environments*, 10thEuropean Workshop on Dependable Computing (EWDC-10), Wiedeń, 1999

[5] Krawczyk H., Proficz J. *KASKADA – multimedia processing platform architecture*, Signal Processing and Multimedia Applications, SciTePress Ateny, 2010

[6] Krawczyk H., Wiszniewski B. *Analysis and testing of distributed software applications*, Baldock: Res.Stud. Press, 1998

[7] Kuck D. J. „High Performance Computing, Challenges for Future Systems", Oxford University Press, 1996

[8] Pressman R. S. *Software engineering: a practitioner's approach*, McGraw-Hill International Editions, 1992

[9] H. Schligloff, Dr. M. Roggenbach *Path Testing*, `http://www.cs.swan.ac.uk/~csmarkus/CS339/dissertations/GregoryL.pdf`

[10] `http://activemq.apache.org/`

[11] Mellanox Technologies *Introduction to InfiniBand*, `www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf`

[12] Andrzej Mączyński, *Oprogramowanie systemów biomedycznych*, `http://biomed.eti.pg.gda.pl/~manczyn/OSB/OSB.html`

[13] *Słownik języka polskiego*, PWN, `http://sjp.pwn.pl/`

[14] *Unix man pages*, `http://unixhelp.ed.ac.uk/CGI/man-cgi?top+1`

[15] *Projekt Trac*, `http://trac.edgewall.org/`