POLITECHNIKA POZNAŃSKA

Wydział Informatyki

Instytut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

Łukasz Antczak (106641)

Paulina Jankowska (106535)

Hanna Nowak (106559)

PLATFORMA DO REALIZACJI I ANALIZY TESTÓW PAMIĘCI PROCEDURALNEJ

Promotor:
dr inż. Miłosz Kadziński

Poznań, 2015

Spis treści

Rozdział 1

Wprowadzenie

1.1 Cel i zakres pracy

Celem niniejszej pracy inżynierskiej jest stworzenie komputerowej platformy do przeprowadzania oraz analizy wyników testów pamięci proceduralnej. System ma służyć do obiektywizacji oceny zaburzeń spostrzegania, doświadczania, psychomotoryki oraz dostrajania do wymagań otoczenia. Docelowym użytkownikiem platformy są jednostki naukowe i kliniczne zajmujące się leczeniem uzależnień, chorób Parkinsona oraz Alzheimera.

Oprogramowanie powinno dawać możliwość przeprowadzenia testów na jednym badanym, na kilku pacjentach z osobna, jak i wykonania serii badań na próbie populacji. Niezbędną cechą aplikacji jest także zapis przebiegu testu wraz z danymi pacjenta i badania. Docelowo podstawowa wersja aplikacji ma zawierać moduły realizacji oraz analizy trzech testów pamięci proceduralnej. Oprogramowanie powinno być jednak zaprojektowane w taki sposób, aby mogło być w przyszłości rozwijane i rozbudowywane o nowe testy, w zależności od potrzeb potencjalnych użytkowników.

Dodatkowym celem projektu jest parametryzacja wykonania testów, aby umożliwić przeprowadzającemu badanie dobór odpowiednich ustawień dla konkretnego pacjenta oraz badanie wpływu różnych czynników na wielkość zaburzeń pamięci proceduralnej.

1.2 Motywacja

Główną motywacją stworzenia platformy jest automatyzacja przeprowadzania testów pamięci proceduralnej tak, aby proces ten stał się bardziej efektywny. Dzięki opracowanemu systemowi możliwa będzie dokładniejsza i szybsza analiza wyników badań. Ich standardowa interpretacja jest bardzo trudna, stąd niezbędna staje się

pomoc technologii.

Uzasadnieniem zbudowania aplikacji jest również brak narzędzia przystosowanego zarówno do przeprowadzenia, jak i analizy wyników badań. Ponadto, dostępne na rynku programy służące tylko do realizacji testu są przestarzałe pod względem proponowanego interfejsu graficznego, sugerowanego sposobu obsługi oraz ograniczonej możliwości parametryzacji. Jednym z przykładów takiego systemu jest platforma PEBL (The Psychology Experiment Building Language) oparta na licencji wolnego i otwartego oprogramowania [PEB].

1.3 Struktura pracy

Niniejsza praca składa się z siedmiu rozdziałów. W rozdziale drugim opisano istotę pamięci proceduralnej, a także idee i znaczenie kliniczne testów zaimplementowanych w systemie. Kolejny rozdział dotyczy specyfikacji wymagań funkcjonalnych i niefunkcjonalnych, które zdefiniowano dla platformy. Rozdział czwarty stanowi opis wykorzystanych technologii, stworzonej struktury projektu systemu oraz zastosowanych wzorców projektowych. W rozdziale piątym szczegółowo wyjaśniono sposób implementacji poszczególnych testów pamięci proceduralnej, ich analizy oraz pozostałej funkcjonalności. Rozdział szósty jest instrukcją korzystania z systemu stworzoną dla przyszłych użytkowników. W ostatnim rozdziale podsumowano zrealizowaną pracę w odniesieniu do postawionych celów i wymagań.

1.4 Podział pracy

Projekt architektury systemu, sposób przechowywania danych oraz ich przepływ został opracowany wspólnie przez wszystkich członków zespołu. Poniżej opisano pozostałe zadania wykonane przez poszczególne osoby.

Łukasz Antczak zaimplementował test śledzenia wirnika. Ponadto, był odpowiedzialny za analizę wyników tego testu. Dodatkowo opracował sposób zapisu przebiegów testów oraz wykorzystując bibliotekę Apache Maven skonfigurował proces automatycznego zarządzania projektem i jego budowy.

Paulina Jankowska była odpowiedzialna za stworzenie graficznego interfejsu użytkownika oraz przepływ sterowania działaniem aplikacji. Zaimplementowała eksport oraz import badań, a także zapis danych do plików XML. Była współodpowiedzialna za wprowadzenie dwujęzyczności systemu.

Hanna Nowak stworzyła moduł analizy statystycznej używany podczas porównywania prób populacji. Ponadto, zaimplementowała przebieg testów Stroopa

oraz sortowania kart z Wisconsin wraz z analizą ich wyników. Była współodpowiedzialna za internacjonalizację platformy.

Rozdział 2

Podstawy teoretyczne

Pamięć proceduralna wraz z pamięcią deklaratywną to rodzaje pamięci długotrwałej, które mają teoretycznie nieograniczoną pojemność i czas przechowywania. Pamięć deklaratywna odnosi się do zapamiętywania faktów, nazw oraz codziennych wydarzeń. Istnieje możliwość bezpośredniego odwołania się do niej przez przypomnienie konkretnych elementów przechowywanych w pamięci.

Pamięć proceduralna, inaczej zwana pamięcią niedeklaratywną, odpowiada za zdolności motoryczne. Obejmuje umiejętności takie jak pływanie, jazda na rowerze, a także pisanie oraz czytanie. Nabywana jest przez doświadczenie i polega na wyuczeniu konkretnej reakcji na bodźce. W konsekwencji wraz z kolejnymi powtórzeniami czynności powinna wzrastać jej wydajność. Nie można odwołać się do tej części pamięci, jak to robimy w przypadku pamięci deklaratywnej, ponieważ dostęp do niej osiągany jest automatycznie. Umiejętności zapisane w pamięci proceduralnej są długotrwałe i nawet długi czas przerwy nie powinien wpływać na jakość ich wykonania [AAF13, Gre08].

W kolejnych rozdziałach omówiono idee, sposoby przeprowadzania i analizy testów pamięci proceduralnej, które zostały uwzględnione w ramach opracowywanej platformy.

2.1 Test sortowania kart z Wisconsin

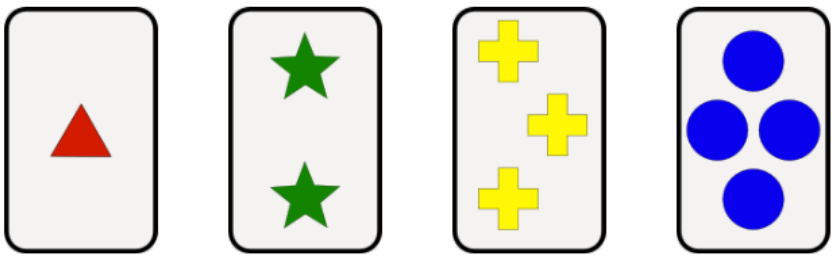
Idea testu

Test sortowania kart z Wisconsin (ang. Wisconsin Card Sorting Test, WCST) został stworzony przez Davida Granta i Esta Berga w 1948r. Na podstawie wyników WCST można stwierdzać stopień uszkodzenia specyficznych części mózgu odpowiadających za pamięć proceduralną i funkcje wykonawcze. Test ten bada zdolność zmiany strategii działania pacjenta w obliczu zmieniających się warunków. Ocenie podlega także

zdolność myślenia abstrakcyjnego, podejmowania decyzji w zależności od środowiska oraz odporność na stres [SSS06].

Do przeprowadzenia testu używane są karty przedstawiające figury geometryczne i różniące się od siebie pod względem kryterium koloru, kształtu elementów oraz ich liczby. Pacjent przez cały czas ma przed sobą cztery karty wzorcowe przedstawiające:

- jeden czerwony trójkąt,
- dwie zielone gwiazdy,
- trzy żółte krzyże,
- cztery niebieskie koła.



Rysunek 2.1: Karty wzorcowe.

Zadaniem badanego jest przyporządkowywanie kolejnych prezentowanych mu kart do kategorii reprezentowanych przez karty wzorcowe. Po każdej karcie pacjent otrzymuje informację, czy jego odpowiedź jest poprawna. Nie jest świadomy przy tym aktualnie obowiązującego kryterium poprawności sortowania kart. W podstawowej wersji testu pierwszym kryterium jest kolor, następnie kształt i na końcu liczba. Po uzyskaniu dziesięciu poprawnych odpowiedzi kryterium zostaje zmienione, o czym badany nie jest informowany.

Początkowo używano symetrycznego rozkładu figur na kartach – jedna na środku, dwie tworzyły przekątną, trzy trójkąt równoramienny, a cztery kwadrat. Jednak to powodowało, że sortowanie wg liczby stawało się łatwiejsze – rozmieszczenie w ten sposób sugerowało ilość kształtów na karcie, przez co w teście brakowało wewnętrznej równowagi. Dlatego też figury zdecydowano się rozmieścić losowo.

Rozwiązywanie testu nie ma limitu czasowego. Trwa on tak długo, aż wszystkie 128 kart zostanie wykorzystanych lub kończy się wcześniej, kiedy każde kryterium wystąpi dwukrotnie [Ste92]. Warto zauważyć, że klasyczna wersja testu była przeprowadzana manualnie. Równoważność wersji manualnej i realizowanej na komputerze jest kontrowersyjna. Ponadto, porównywanie wyników WCST pomiędzy różnymi wersjami testu, które w jakikolwiek sposób zmieniają przebieg lub parametry wykonywania badania nie jest miarodajne.

Modyfikacje testu

Na przestrzeni lat powstało wiele modyfikacji WCST. Jednym z wariantów testu jest modyfikacja Nelsona z 1976r. (ang. Modified Card Sorting Test, MCST). Spośród wszystkich kart bodźcowych usunięto te, które uniemożliwiały jednoznaczne przyporządkowanie podjętej przez badanego decyzji do kryterium wyboru (koloru, liczby lub kształtu). Ponadto, zmiana kryterium następuje tutaj po sześciu poprawnych odpowiedziach, o czym pacjent jest informowany. Zmiany takie sprawiają, że wyliczone parametry stają się bardziej rzetelne i łatwiejsze do analizy. Wielu innych naukowców rozwijających WCST modyfikowało liczbę dobrych odpowiedzi koniecznych do zmiany kategorii, czy też kolejność kryteriów dopasowania [SSS06].

Analiza wyników

Wynikiem przeprowadzonego badania jest zapis wszystkich odpowiedzi pacjenta w kontekście kryterium poprawności dopasowania. Na podstawie tych danych ocenia się wskaźniki wykonania testu. Wskaźniki, które posiadają interpretację kliniczną zostały przedstawione poniżej:

- liczba błędów ogółem (ang. total response errors) – liczba wszystkich odpowiedzi niezgodnych z aktualnym kryterium;
- liczba zaliczonych kategorii (ang. number of categories completed) – liczba sekwencji określonej liczby poprawnych odpowiedzi (zaliczenia kategorii); dla standardowej wersji testu wynosi ona 10, dla modyfikacji Nelsona – 6; określa ogólną poprawność wykonania testu;
- próby przeprowadzone do momentu zaliczenia pierwszej kategorii (ang. trials to complete first category) – liczba udzielonych odpowiedzi do momentu zaliczenia pierwszej kategorii; charakteryzuje umiejętność koncentracji, szybkość rozpoznania zasady testu;
- odpowiedzi perseweracyjne (ang. perseverative responses) – odpowiedzi, które uwzględniają persewerowane kryterium, ale nie muszą być błędne (będą zawsze błędne dla modyfikacji Nelsona, w której eliminuje się karty pasujące do karty wzorcowej pod względem więcej niż jednego kryterium); charakteryzują one trudności w zmianie nastawienia poznawczego;
- błędy perseweracyjne (ang. perseverative errors) – odpowiedzi, które uwzględniają persewerowane kryterium i są niezgodne z aktualnym kryterium;

- błędy niepersewerac yjne (ang. nonperseverative errors) – liczba błędnych odpowiedzi, które nie uwzględniają kryterium persewerowanego; błędy te dotyczą problemów myślenia abstrakcyjnego i kategoryzowania niezwiązanych z nastawieniem poznawczym (jak w przypadku błędów perseweracyjnych);

- procent błędów perseweracyjnych (ang. percent perseverative errors) – procent błędów perseweracyjnych z liczby przeprowadzonych prób; jest miarą pozwalającą na porównywanie testów różniących się liczbą przeprowadzonych prób;

- porażka w utrzymaniu nastawienia (ang. failure to maintain set) – liczba błędów dokonanych między piątą dobrą odpowiedzią z rzędu a zaliczeniem kategorii; charakteryzuje umiejętność koncentracji uwagi;

- procent odpowiedzi pojęciowych (ang. percent conceptual level responses) – procent poprawnych odpowiedzi występujących w serii co najmniej trzech z rzędu w stosunku do liczby przeprowadzonych prób; jest miarą efektywności kategoryzowania;

- uczenie się uczenia (ang. learning to learn) - miara spadku całkowitej liczby odpowiedzi potrzebnych do zaliczania kolejnych kategorii; parametr jest obliczany po ukończeniu trzech lub więcej kategorii i charakteryzuje umiejętność uczenia się jako poprawę szybkości zaliczania kolejnych kategorii.

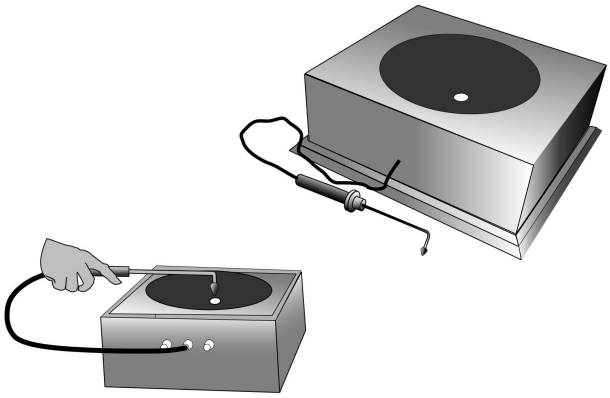
Ocena błędów perseweracyjnych jest kontrowersyjna i dostarcza najwięcej problemów. Jest to jednocześnie jeden z kluczowych elementów testu – persewerac ja może wskazywać na różne schorzenia pacjenta. Bardzo trudno odróżnić persewerację, czyli uporczywą, niezmienną odpowiedź, mimo otrzymywania informacji wskazującej na zmianę strategii wyboru kryterium dopasowania kart, od przypadkowej pomyłki. W WCST zdarza się także, że z powodu niejednoznaczności kryterium nie da się określić momentu, w którym pacjent otrzymuje informację o niepoprawności przyjętej przez niego strategii. Modyfikacja Nelsona niweluje wspomniane problemy poprzez usunięcie z zestawu kart bodźcowych niejednoznacznych pod względem kryterium dopasowania [Jaw02].

2.2 Test śledzenia wirnika

Idea testu

Test śledzenia wirnika ma na celu badanie sprawności psychomotorycznej oraz zdolności uczenia się proceduralnego. Jest bardzo użyteczny do oceny kontroli ruchu

(ang. motor control function). W początkowych latach test śledzenia wirnika wykonywany był na specjalnych przyrządach. Badana osoba miała za zadanie utrzymać wskaźnik nad punktem umieszczonym na rotacyjnie poruszającej się płycie urządzenia (patrz Rysunek 2.2). Największe znaczenie ma koordynowanie ruchów w trakcie wykonywania zadania [Ada52, Amm55].



Rysunek 2.2: Urządzenie służące do wykonywania testu śledzenia wirnika.

Wersja komputerowa testu polega na utrzymaniu kursora myszki nad kołem poruszającym się po okręgu. Zadanie pacjenta polega na jak najdłuższym utrzymaniu wskaźnika na tym punkcie.

Analiza wyników

W teście śledzenia wirnika analizie podlega przede wszystkim czas utrzymania kursora w obrębie punktu oraz amplituda drżeń kursora. Dodatkowymi parametrami obliczanymi podczas analizy jest średnia oraz chwilowa odległość od brzegu punktu, a także od jego środka. Podział okręgu na części dodatkowo pozwala zauważyć zależności pomiędzy fragmentami okręgu, a wydajnością wykonywania zadania. Ważnym elementem analizy jest uczenie się motoryczne, które można zaobserwować po kilku próbach wykonania zadania. Dokładność w wykonaniu zadania ukazuje zaburzenia koordynacji motorycznej oko-ręka.

2.3 Test Stroopa

Idea testu

W 1935 roku John Ridley Stroop stworzył procedurę nazywania koloru, w którym napisane jest słowo oznaczające inny kolor. Procedura ta obrazuje umiejętność rozwiązywania zadań polegających na rozróżnieniu konfliktowych informacji obrazowych i semantycznych. Pierwotna wersja testu składała się z czterech części:

- odczytanie nazw kolorów napisanych czarną czcionką,
- odczytanie nazw kolorów napisanych innym kolorem niż wskazuje tekst,
- nazwanie kolorów przedstawionych kwadratów,
- nazwanie kolorów, którymi napisane są słowa oznaczające inny kolor.

Istnieją różne modyfikacje standardowej wersji testu. Jedną z nich jest ograniczenie tylko do czwartej części wersji podstawowej. Pokazywane jest jedno słowo napisane innym kolorem niż wskazuje na to tekst oraz możliwe odpowiedzi. Liczba odpowiedzi zależy od liczby użytych w teście kolorów. Przejście do kolejnego słowa następuje dopiero po udzieleniu poprawnej odpowiedzi. Test Stroopa stosowany jest jako wskaźnik zdolności koncentracji i podzielności uwagi, jakości funkcjonowania poznawczego oraz kontroli wykonawczej.

Analiza wyników

W analizie testu Stroopa największe znaczenie ma czas wykonania zadania. Gdy nazwa nie zgadza się z kolorem, udzielenie odpowiedzi na pytanie zabiera więcej czasu i istnieje większe prawdopodobieństwo popełnienia błędu niż normalnie. Czynność ta nie jest automatyczna, wymaga zastanowienia się. Dodatkowo można analizować średnią liczbę błędów popełnionych w czasie realizacji zadania [Str35].

Rozdział 3

Specyfikacja wymagań

3.1 Wymagania funkcjonalne

Wymaganiami funkcjonalnymi nazywamy określenie wszystkich usług i możliwości, które system powinien oferować, a także jak powinien zachować się w określonych sytuacjach. Wymagania te zależą od typu wytwarzanego oprogramowania i oczekiwań potencjalnych użytkowników. Przeważnie tworzy się je w języku naturalnym, aby były zrozumiałe dla klienta [Som11].

Dla platformy do przeprowadzania i analizy testów pamięci proceduralnej zdefiniowane następujące wymagania funkcjonalne:

1. Użytkownik powinien mieć możliwość wyboru i zmiany folderu roboczego, w którym zapisywane będą pliki z przebiegami badań. System powinien wówczas utworzyć w wybranym katalogu niezbędne do działania podkatalogi. Oparcie wyników realizacji i analizy testów na systemie plików, nie zaś na zewnętrznej bazie danych, jest wymaganiem sformułowanym przez docelową grupę użytkowników.

2. Użytkownik powinien mieć możliwość wyboru jednego lub kilku z dostępnych testów do przeprowadzenia. Pierwsza wersja platformy powinna oferować komputerowe realizacje następujących testów:

   - sortowania kart z Wisconsin,
   - śledzenia wirnika,
   - Stroopa.

3. Użytkownik powinien mieć możliwość dostosowania testu do potrzeb badania. Każdy test powinien posiadać specyficzną dla siebie konfigurację, w której zdefiniowane będą charakterystyczne parametry.

4. System powinien umożliwić wprowadzenie informacji o badanym z wykorzystaniem dedykowanego formularza.

5. System powinien umożliwić wybór oraz ewentualną edycję danych pacjenta, który już wcześniej miał przeprowadzone testy, w celu przyspieszenia wykonywania badania.

6. Użytkownik powinien mieć możliwość zapisania badania wraz z datą i miejscem jego przeprowadzenia.

7. Użytkownik powinien mieć możliwość przeprowadzenia i zapisania serii badań, np. dla danej próby populacji.

8. System powinien umożliwić automatyczną analizę wybranego testu:

    a) test sortowania kart z Wisconsin – wyświetlenie wyliczonych wartości następujących wskaźników:
        - liczba błędów ogółem,
        - liczba zaliczonych kategorii,
        - próby przeprowadzone do momentu zaliczenia pierwszej kategorii,
        - odpowiedzi perseweracyjne,
        - błędy perseweracyjne,
        - błędy niepersewerac yjne,
        - procent błędów perseweracyjnych,
        - porażka w utrzymaniu nastawienia,
        - procent odpowiedzi pojęciowych,
        - uczenie się uczenia.

    b) test śledzenia wirnika:
        - analiza FFT i zobrazowanie jej wyników na wykresie;
        - odtworzenie przebiegu testu on-line;
        - porównanie fragmentów okręgu pod względem procentowego utrzymania kursora w obrębie punktu oraz średniej odległości od punktu;
        - porównanie kolejnych serii testu;
        - wykres średniej odległości od punktu;
        - wyliczenie wartości procentu czasu spędzonego w obrębie punktu, średniej odległości od środka punktu oraz średniej odległości od brzegu punktu.

c) test Stroopa – wyświetlenie wyliczonych wartości następujących parametrów:

- średni czas potrzebny do udzielenia poprawnej odpowiedzi,
- średnia liczba błędów popełnionych do udzielenia poprawnej odpowiedzi,
- liczba błędów ogółem,
- całkowity czas wykonania.

9. Platforma powinna umożliwić analizę porównawczą dwóch testów tego samego rodzaju.

10. Użytkownik powinien mieć możliwość analizy dwóch prób populacji. Dodatkowo należy udostępnić opcję zapisu stworzonych par osobników jako projekt, by w przyszłości móc je odtworzyć.

11. Aplikacja powinna umożliwić sprawdzenie, czy rozkład badanej cechy jest rozkładem normalnym za pomocą testu Kołmogorowa-Smirnowa, a także przeprowadzenie testów parametrycznych (testy t-Studenta) lub nieparametrycznych (test Wilcoxona lub test U Manna-Whitneya) dla weryfikacji istotności statystycznej różnic wybranych parametrów lub wskaźników dla prób z populacji.

12. Użytkownik powinien mieć możliwość eksportu/importu wyników realizacji badań z/do aplikacji, w celu ich wymiany z innymi użytkownikami korzystającymi z tej platformy.

13. System powinien umożliwiać filtrowanie danych, przyspieszające proces wyboru badań do analizy oraz eksportu.

3.2 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne dotyczą ograniczeń nałożonych na system lub na funkcje oferowane przez niego. Skupiają się one na takich właściwościach aplikacji jak bezpieczeństwo, niezawodność, czas reakcji, przenaszalność czy użyteczność. Wymagania te są ważniejsze od wymagań funkcjonalnych, gdyż ich niespełnienie może spowodować bezużyteczność systemu [Som11].

Dla opracowanej platformy zdefiniowano następujące wymagania niefunkcjonalne:

1. Umożliwienie korzystania z oprogramowania niezależnie od systemu operacyjnego, mocy obliczeniowej oraz rozdzielczości ekranu.

2. Zapewnienie intuicyjnego interfejsu użytkownika.

3. Dostosowanie systemu do dalszego rozwoju, dodania nowych testów.

4. Stworzenie jednolitego formatu danych dla eksportu i importu badań.

5. Umożliwienie korzystania z systemu w przypadku braku dostępu do Internetu.

6. Zapewnienie działania oprogramowania bez używania zewnętrznej bazy danych.

7. Wsparcie dla języka polskiego i angielskiego.

3.3 Parametryzacja testów pamięci proceduralnej

Jednym z celów pracy jest zmodyfikowanie istniejących testów pamięci proceduralnej oraz wprowadzenie nowych parametrów konfiguracyjnych, aby badający mógł dostosować przebieg testu do swoich potrzeb. Taka charakterystyka w znaczący sposób zwiększa użyteczność systemu.

3.3.1 Test sortowania kart z Wisconsin

Dla testu sortowania kart z Wisconsin uwzględniono następujące parametry konfiguracyjne:

1. Wybór pomiędzy standardową wersją testu, a opisaną w Rozdziale 2.1 modyfikacją Nelsona.

2. Ustawienie, czy informacja dotycząca zmiany kryterium po ustalonej liczbie poprawnych odpowiedzi ma zostać udzielona badanemu, a także o tym, czy w momencie tej zmiany pacjent ma zostać poinformowany.

3. Losowa lub narzucona jak w podstawowej wersji (kolor, kształt, liczba) kolejność kryteriów dopasowania.

4. Opcja pokazywania badanemu ostatniej jego odpowiedzi z informacją, bądź nie, o jej poprawności.

5. Informacja o liczbie poprawnych odpowiedzi z rzędu.

3.3.2 Test śledzenia wirnika

Komputerowa implementacja testu śledzenia wirnika zapewnia bardzo precyzyjny pomiar przebiegu testu oraz umożliwia wprowadzenie wielu parametrów konfiguracyjnych:

1. Zdefiniowanie wielkości śledzonego punktu.
2. Ustawienie promienia okręgu, po którym porusza się punkt.
3. Ustawienie czasu trwania jednego pełnego obrotu punktu.
4. Wybór kierunku poruszania się punktu – zgodny lub przeciwny do ruchu wskazówek zegara.
5. Wybór miejsca na okręgu, w którym ma się rozpocząć test.
6. Ustalenie liczby serii oraz liczby okrążeń w serii.
7. Ustawienie, czy pacjent ma być poinformowany o kierunku poruszania się punktu przed rozpoczęciem testu.
8. Opcja zmiany koloru kółka w zależności, czy kursor jest w jego obrębie.

3.3.3 Test Stroopa

Dla testu Stroopa udostępnione zostały następujące parametry konfiguracji:

1. Wybór wersji testu:
   - czytanie kolorów napisanych czarnym drukiem,
   - nazwanie koloru, w którym napisane jest słowo oznaczające inny kolor.
2. Zdefiniowanie sposobu obsługi testu: za pomocą myszki lub klawiatury.
3. Wybór liczby prób oraz kolorów spośród dostępnych.
4. Odpowiedzi na pytania napisane czcionką czarną lub kolorową.
5. Możliwość eliminacji słów napisanych tym samym kolorem, na który wskazuje treść.
6. Możliwość przeprowadzenia serii próbnej.

Rozdział 4

Architektura aplikacji

4.1 Opis wykorzystanych technologii

Platforma do realizacji i analizy testów pamięci proceduralnej została stworzona w języku **JAVA** z wykorzystaniem Java Development Kit w wersji 1.8 przy pomocy zintegrowanego środowiska programistycznego (ang. Integrated Development Environment, IDE) IntelliJIDEA 13.1 Community Edition firmy JetBrains. Decyzję o wyborze tego języka podjęto na podstawie następujących przesłanek:

- obiektowość języka JAVA, która zapewnia elastyczne możliwości rozwoju oprogramowania;
- wieloplatformowość języka JAVA będąca spełnieniem wymagania niefunkcjonalnego dotyczącego działania programu w prawidłowy sposób niezależnie od systemu operacyjnego;
- szeroka gama bibliotek oraz narzędzi otwartego oprogramowania (ang. open source) oferowanych dla tego języka;
- doświadczenie wszystkich członków zespołu w programowaniu w języku JAVA.

Wybór środowiska programistycznego podyktowany był następującymi faktami:

- intuicyjna i prosta obsługa IntelliJIDEA;
- dobra znajomość środowiska programistycznego IntelliJIDEA przez wszystkich członków zespołu;
- bogata oferta funkcjonalnych wtyczek dla IDE.

W celu stworzenia interfejsu graficznego użytkownika wykorzystano bibliotekę **JavaFX 8**. Umożliwia ona oddzielenie interfejsu graficznego od warstwy logiki systemu. Dostarcza wiele komponentów, które mogą być dostosowywane za pomocą

arkuszy stylów CSS (ang. Cascading Style Sheets). Graficzny interfejs użytkownika (ang. Graphical User Interface, GUI) w JavaFX można tworzyć w dwojaki sposób: poprzez kod w Javie albo za pomocą języka opartego na uniwersalnym języku znaczników XML – FXML. Dobrym narzędziem do interaktywnego tworzenia graficznego interfejsu, bez pisania kodu, jest aplikacja JavaFX Scene Builder, która generuje plik z rozszerzeniem .fxml. JavaFX umożliwia również rysowanie poprzez Canvas API (ang. Application Programming Interface) oraz tworzenie wykresów. Jednym z bardzo wygodnych mechanizmów oferowanych przez JavaFX jest obsługa zdarzeń (ang. events handling). Dzięki tej funkcjonalności w łatwy sposób można zdefiniować akcje realizowane przez program w momencie wykrycia określonej aktywności użytkownika aplikacji.

Zarządzanie procesem budowania aplikacji odbywa się z wykorzystaniem **Apache Maven**. Narzędzie to automatyzuje cały proces kompilacji, korzystając z pliku konfiguracji stworzonej przez zespół programistyczny. Warto podkreślić fakt, iż Maven przedkłada przyjęcie konwencji nad konfigurację (and. Convention over Configuration, CoC). Oznacza to, że nie jest wymagane jawne ustawianie wszystkich cech pracy narzędzia. Konfiguracja odbywa się poprzez plik określający sposób budowy oprogramowania tzw. POM (ang. Project Object Model), który jest dokumentem XML. Zdefiniowane w nim są wszystkie potrzebne do kompilacji zależności, czyli biblioteki, bez których program nie jest zdolny do działania. Niewątpliwą zaletą narzędzia jest fakt, że zależności są ładowane w sposób automatyczny z centralnego repozytorium. Eliminuje to proces żmudnego pobierania bibliotek z różnych stron internetowych i nie wymaga dodatkowej konfiguracji projektu w danym IDE. Kolejną z interesujących cech Apache Maven jest bogata oferta wtyczek. Przykładem takiej wtyczki jest *maven-assembly-plugin* ułatwiający w znaczący sposób przygotowanie wykonywalnego pliku JAR (ang. Java ARchive) wraz ze wszystkimi potrzebnymi do jego poprawnego działania zasobami [MAV].

W celu poźniejszej analizy przebiegu testu wymagane jest zapisywanie ustalonych danych dotyczących różnych zdarzeń mających miejsce podczas wykonywania testu przez pacjenta. Zapis tych danych odbywa się asynchronicznie za pomocą mechanizmu logów i jest ukryty dla użytkownika. Log to tzw. dziennik zdarzeń zapisywanych przez aplikację. Biblioteką, która udostępnia taką funkcjonalność jest **log4j** wydawana przez Apache Foundation. Korzystanie z tego narzędzia jest bardzo proste i efektywne Konfiguracja działania tej biblioteki może odbywać się na trzy różne sposoby: poprzez plik właściwości (*log4j.properties*), przez dedykowany plik XML oraz bezpośrednio z poziomu kodu aplikacji. w platformie zdecydowano się wykorzystać trzeci sposób, ponieważ daje on możliwość zmiany sposobu działa-

nia log4j w trakcie działania aplikacji, co jest niezwykle przydatne przy zapisywaniu przebiegu testów (każde wykonywane badanie zapisywane jest do innego pliku).

Ze względu na wymagania funkcjonalne w aplikacji zrezygnowano z używania zewnętrznej bazy danych. Wymusiło to opracowanie sposobu ewidencjonowania informacji o pacjentach tak, aby nie było potrzeby kilkukrotnego wprowadzania tych samych danych. Ewidencji powinny podlegać również przeprowadzone badania, w celu ułatwienia późniejszego wyboru ich do analizy czy eksportu. Ostatecznie zdecydowano się przechowywać dane w plikach XML. Do realizacji wymienionych celów wykorzystano technologię **JAXB** (ang. Java Architecture for XML Binding), która pozwala na konwertowanie obiektów w języku Java na ich odpowiedniki w pliku XML i odwrotnie. Nie wymaga to głębszej znajomości sposobu przetwarzania dokumentów XML pozwalając jednocześnie na efektywny zapis i odczyt danych.

Do przeprowadzenia analizy statystycznej użyto zewnętrznej biblioteki **Apache Commons Math**. Zawiera ona implementacje funkcji, których wykorzystanie jest konieczne do przeprowadzenia wszystkich testów statystycznych użytych w aplikacji:

- Kołmogorova-Smirnova,
- t-Studenta dla prób zależnych i niezależnych,
- U Manna-Whitneya,
- Wilcoxona.

Wspomniania biblioteka udostępnia również szereg funkcji matematycznych związanych między innymi z algorytmem Szybkiej Transformaty Fouriera (ang. Fast Fourier Transform, FFT) wykorzystywanym w module analizy testu śledzenia wirnika.

W celu efektywnego tworzenia oprogramowania skorzystano z rozproszonego systemu kontroli wersji **Git**. Pozwala on na jednoczesną pracę nad projektem wielu osób, pomagając w łączeniu zmian dokonanych w plikach źródłowych. Główną jego zaletą jest istnienie repozytorium zdalnego i lokalnych, więc w razie utracenia projektu przez serwer, pozostaje on u klientów. System kontroli wersji pozwala także przeglądać zmiany wprowadzone przez konkretnych programistów, a także przywracać projekt do jego wcześniejszych postaci.

Jednym z wymagań niefunkcjonalnych jest stworzenie interfejsu użytkownika w dwóch językach – angielskim i polskim. Problem wielojęzyczności w działaniu aplikacji nazywany jest **internacjonalizacją** (ang. internationalization, i18n). w projekcie wsparcie dwóch języków zostało zrealizowane za pomocą pakietu zasobów (ang. resource bundle), czyli rozwiązaniu, które jest wbudowane w JDK. Cały proces definiowania pliku mapującego komunikaty podlegające tłumaczeniu jest wspierany przez wtyczkę IntelliJIDEA. Warto dodać, iż taki stan rzeczy powoduje, że dodanie

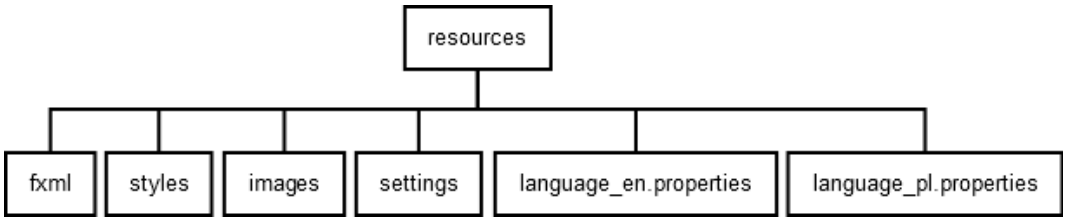
kolejnego języka wymaga jedynie dołączenia kolejnego pliku z zasobami językowymi.

4.2 Szczegółowy opis struktury projektu

Projekt systemu składa się z pakietów klas Javy oraz z zasobów (ang. resources). w kolejnych rozdziałach omówiono najważniejsze z punktu widzenia działania oprogramowania elementy projektu.

4.2.1 Zasoby

Organizacja zasobów w ramach projektu ma charakter struktury katalogów (patrz Rysunek 4.1).



Rysunek 4.1: Organizacja zasobów.

Najważniejszym spośród nich jest folder ***fxml***, zawierający dokumenty FXML opisujące następujące widoki interfejsu graficznego:

- główne okno aplikacji,
- okna konfiguracyjne badań (formularz pacjenta, ustawienia parametrów testów, itp.).

Znajdują się w nim również niestandardowe komponenty stworzone na potrzeby aplikacji:

- *filterPane.fxml* służący do przedstawienia filtrów, niezbędnych do szybszego wyszukiwania umieszczonych w tabeli badań;
- *orderedTable.fxml* stanowiący tabelę z możliwością ustawiania kolejności elementów.

Dzięki dokumentom FXML możliwe jest zarządzanie widokami niezależnie od logiki systemu. Każdy z plików zawiera nazwę swojego kontrolera, który jest zaimplementowany jako klasa Javy. Ponadto, komponenty mają jasno określone położenie, a także rozmiar. Identyfikator każdego z nich stanowi powiązanie z polami, a opcjonalnie zdefiniowane akcje z metodami klasy kontrolera. Listing 4.1 przedstawia fragment pliku FXML opisującego widok okna wyboru pacjentów.

Listing 4.1: Fragment pliku FXML dotyczącego okna wyboru pacjentów.

```
<Pane prefHeight="375.0" prefWidth="764.0" xmlns="http://javafx.com
   /javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="
   put.testModule.window.controller.PatientsChooseController">
   <children>
       <Label fx:id="savedPatientLbl" layoutX="14.0" layoutY="25.0"
           prefHeight="17.0" prefWidth="153.0" />
       <TableView fx:id="savedPatientsTbl" layoutX="14.0" layoutY="
          48.0" prefHeight="200.0" prefWidth="347.0"/>
       <Button fx:id="choosePatientBtn" layoutX="366.0" layoutY="
          72.0" mnemonicParsing="false" onAction="#choosePatient"
          prefHeight="25.0" prefWidth="32.0" text="&gt;" />
       ...
 </children>
 </Pane>
```

Kontrolerem ekranu dotyczącego wyboru pacjentów jest klasa *PatientChooseController*. Najwyższym w hierarchii komponentów jest panel o zdefiniowanym dokładnie rozmiarze. Na nim umieszczone są kontrolki służące interakcji z użytkownikiem, między innymi przycisk o identyfikatorze *choosePatientBtn*. Po kliknięciu na niego ma zostać wywołana metoda *choosePatient*.

W katalogu ***styles*** znajdują się arkusze stylów CSS. Możliwość ustawienia styli sprawia, że widoki stosowane w oprogramowaniu są spójne i bardziej przyjazne użytkownikowi. Arkusze stworzone w projekcie konfigurują wygląd kontrolek używanych w interfejsie graficznym użytkownika, a także wygląd wykresów używanych podczas analizy testów. Listing 4.2 przedstawia fragment stylu określającego podrodzinę czcionki, kolor i grubość obramowania oraz wygląd kursora, gdy znajdzie się na konfigurowanych kontrolkach.

Listing 4.2: Fragment stylu styles.css definiujący wygląd przycisków i rozwijanych list występujących w aplikacji.

```
.button, .combo-box{
    -fx-cursor: hand;
    -fx-border-width: 1;
    -fx-border-color: #395bae;
    -fx-font-weight: bold;
}
```

W folderze ***images*** znajdują się obrazy wykorzystywane w aplikacji; są to w większości graficzne reprezentacje kart używanych w teście sortowania kart z Wisconsin.

Katalog ***settings*** zawiera pliki ustawień domyślnych:

- *settings.xml*, zapisywany jako domyślne ustawienia użytkownika dotyczące wybranego folderu roboczego, a także ustawionego języka aplikacji (patrz Listing 4.3); podczas uruchomiania platformy podlega on konwersji na obiekt klasy *Settings*, zaś podczas wyłączania systemu operacji odwrotnej;
- pliki XML z ustawieniami konfiguracji poszczególnych testów; użytkownik może dostosować każdy test do swoich potrzeb i zachować najczęściej powtarzający się zestaw ustawień.

Listing 4.3: Przykładowa zawartość pliku *settings.xml*.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<settings>
    <defaultDir>D:\Example</defaultDir>
    <defaultLanguage>POLISH</defaultLanguage>
</settings>
```

Do zasobów zaliczane są również pliki odpowiadające za **internacjonalizację** oprogramowania. Zorganizowane są one w jeden pakiet zasobów (patrz Rozdział 4.1). W jednym z plików znajdują się polskie znaczenia nazw zastosowanych w aplikacji, w drugim ich angielskie odpowiedniki. Każdy z wyrazów jest zapisany w postaci `klucz = wartość` (patrz Listing 4.4), dzięki czemu w zależności od ustawionego języka pobierane słowo identyfikowane jest po kluczu.

Listing 4.4: Fragment pliku language_en.properties.

```
TEST_MODULE = Tests Module
ANALYSIS_MODULE = Analysis Module
CHANGE_WORKSPACE = Change workspace
IMPORT_EXAM = Import examination
EXPORT_EXAM = Export examination
```

4.2.2 Struktura pakietów

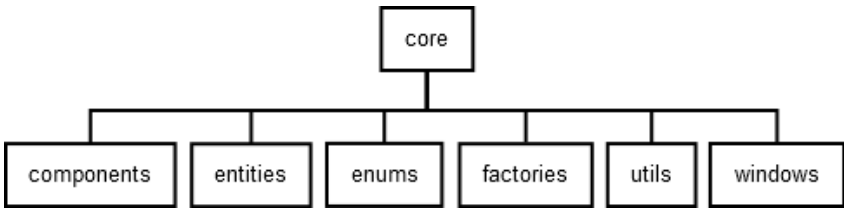
Najwyższy w hierarchi aplikacji pakiet **put** dzieli się na 4 części:

- pakiet **core** zawierający klasy używane w innych pakietach,
- pakiet **testModule** stanowiący moduł testów pamięci proceduralnej,
- pakiet **dataAnalysisModule** z pakietami klas dotyczących analizy badań,
- pakiet **mainModule** z klasą *MainApplication*, w której następuje inicjalizacja aplikacji, a także z klasami odpowiedzialnymi za import i eksport danych.

W tym rozdziale szczegółowo opisano rolę trzech pierwszych z przytoczonych powyżej pakietów.

Pakiet core

W pakiecie core umieszczone zostały kluczowe dla projektu, wykorzystywane w innych modułach klasy. Dla przejrzystości zostały podzielone na mniejsze pakiety (patrz Rysunek 4.2).



Rysunek 4.2: Struktura pakietu core.

W pakiecie **components** znajdują się kontrolery niestandardowych komponentów, stworzonych jako dokument FXML i wspomnianych w poprzednim rozdziale, lub klasy rozszerzające funkcjonalność podstawowych kontrolek.

Pakiet **entities** służy do przechowywania klas, których obiekty podlegają zapisowi do pliku XML. Najważniejsze z nich to klasy reprezentujące badanie, pacjenta oraz serię badań.

W pakiecie **enums** można znaleźć definicje typów wyliczeniowych takich jak dostępne języki czy płcie. Służą one przede wszystkim poprawie przejrzystości kodu i ułatwiają dodanie nowej wartości.

Pakiet **factories** zawiera dwie klasy:

- *ColumnsFactory* – „fabrykę" kolumn dla tabel; w związku z faktem, że w aplikacji informacje są przechowywane często w tabelach, klasa ta zabezpiecza przed powieleniem kodu;
- *TestFactory*, dzięki której można pobrać listę instancji testów dostępnych na platformie.

Pakiet **utils** zawiera kluczowe dla działania aplikacji klasy:

- *LanguageController* – odpowiada za internacjonalizację; dzięki tej klasie możliwe jest wsparcie aplikacji dla języka polskiego i angielskiego;
- *ExamPerformance* – klasa, której występuje tylko jedna instancja w aplikacji; służy do przechowywania m.in. list aktualnie wybranych pacjentów i testów podczas poszczególnych etapów przeprowadzania badania;

- *Settings* – klasa ustawień, która zawiera ścieżki najważniejszych katalogów dla aplikacji: wybranego przez użytkownika folderu roboczego, utworzonych w nim folderu *tests* dla przechowywania przebiegów testów oraz pliku *main.xml* zawierającego skonwertowane obiekty zapisanych pacjentów; klasa ta jest odpowiedzialna za utworzenie wszystkich koniecznych podkatalogów i plików.

- *XmlManager* – klasa zawierająca metody zapisu i odczytu obiektów do/z pliku XML; korzysta z funkcjonalności JAXB, zabezpiecza przed powieleniem kodu (odczyt pacjentów czy serii występuje w kilku miejscach działania aplikacji).

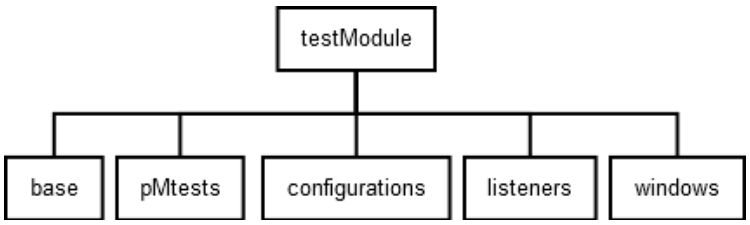
W pakiecie **windows** zdefiniowane są dwa pakiety klas:

- pakiet controller – zawiera abstrakcyjną klasę, z której dziedziczą wszystkie kontrolery widoków FXML;

- pakiet view – składa się z abstrakcyjnych klas, z których dziedziczą klasy odpowiedzialne za wyświetlenie okien bez użycia dokumentów FXML oraz klasy widoków ładujących pliki FXML.

Pakiet testModule

Pakiet ten zawiera klasy odpowiedzialne za realizację testów pamięci proceduralnej, a także cały proces prowadzący do przeprowadzenia badania. Rysunek 4.3 przedstawia podział modułu na pakiety, które zostaną opisane w dalszej części rozdziału.



Rysunek 4.3: Struktura pakietu testModule.

W pakiecie **base** znajdują się klasy abstrakcyjne, z których dziedziczą wszystkie ziamplementowane testy, konfiguracje testów oraz kontrolery ich widoków FXML.

Pakiet **listeners** zawiera interfejsy, implementowane przez obserwatorów. Jeden z nich dotyczy nasłuchiwania zakończenia testu, drugi wypełnienia formularza pacjenta.

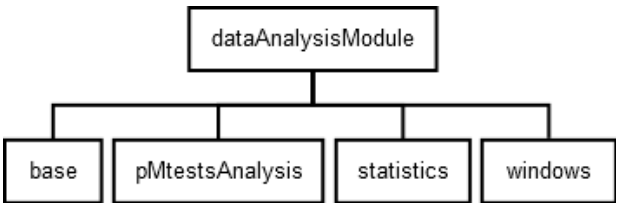
W pakiecie **configurations** umieszczone zostały klasy odpowiadające za parametryzację zaimplementowanych testów. Każda z klas zawiera oddzielny dokument FXML służący zobrazowaniu dostępnych ustawień. W pakiecie znajdują się również klasy kontrolerów wspomnianych widoków.

Pakiet **pMTests** podzielony jest na pakiety klas dotyczących zaimplementowanych w aplikacji testów pamięci proceduralnej. Szczegóły implementacyjne dotyczące każedego z testów znajdują się w Rozdziale 5.

W pakiecie **windows** zdefiniowane są dwa pakiety: controller i view. Pierwszy dotyczy kontrolerów okien, a drugi klas wywołujących pliki FXML.

Pakiet dataAnalysisModule

W pakiecie znajdują się klasy, które są wykorzystywane w analizie wyników badań. Pakiet został podzielony w taki sposób, aby oddzielić klasy podstawowe od odpowiedzialnych za analizę statystyczną czy analizę poszczególnych testów (patrz Rysunek 4.4).



Rysunek 4.4: Struktura pakietu dataAnlysisModule.

Pakiet **base** zawiera klasy abstrakcyjne dla wszystkich rodzajów analizy oferowanych przez aplikację:

- analiza pojedynczego testu,
- analiza dwóch testów,
- porównanie dwóch prób populacji,
- analiza różnic między populacjami.

W zależności od wybranej wersji analizy parametrami konstruktorów klas może być różna liczba obiektów reprezentujących badania.

W pakiecie **pMtestsAnalysis** znajdują się pakiety, oddzielne dla każdego testu, zawierające klasy pomocnicze oraz implementujące wszystkie warianty analizy testów.

Pakiet **statistics** zawiera implementacje testów statystycznych oraz wyświetlania ich wyników. Znajdują się w nim także abstrakcyjne klasy, których rozszerzenia służą do przechowywania informacji o parametrach, na których można przeprowadzić testy statystyczne; są to klasy *AbstractResult* i *AbstractPopulationResult* – odpowiednio dla wyniku analizy pojedynczego testu oraz wyników próby populacji.

W pakiecie **windows** znajdują się klasy reprezentujące okna wyboru badań, na których ma zostać przeprowadzona analiza.

4.3 Zastosowane wzorce projektowe

Wzorzec projektowy to uniwersalne rozwiązanie często pojawiających się problemów programistycznych oraz wygodny sposób na ponowne wykorzystanie kodu zorientowanego obiektowo w różnych projektach i przez różnych programistów. Ideą wzorców projektowych jest zestawienie popularnych interakcji pomiędzy obiektami tak, by przy pojawieniu się podobnego problemu wykorzystać to samo rozwiązanie. W projekcie zastosowano dwa popularne wzorce projektowe: **model-widok-kontroler** (ang. Model-View-Controller, MVC) oraz **Singleton**.

Pierwszy z nich odseparowuje logikę systemu od interfejsu użytkownika. Składa się z trzech elementów: modelu, widoku i kontolera. Warstwa widoku odpowiada za prezentację danych użytkownikowi, model definiuje sposób przechowywania danych, natomiast kontroler odpowiada za logikę i interakcję z użytkownikiem. W aplikacji wzorzec ten zastosowany jest podczas implementacji kolejnych okien konfiguracyjnych przed rozpoczęciem badania.

Wzorzec Singleton należy do grupy wzorców kreacyjnych. Jego celem jest ograniczenie tworzenia instancji danej klasy do jednej. Często występuje w sytuacjach, kiedy każdy element systemu powinien mieć dostęp do tego samego obiektu tej klasy. Wzorzec Singleton implementuje się jako klasę zawierającą statyczną metodę, która sprawdza istnienie instancji tej klasy – jeśli instancja nie istneje, tworzy ją i docelowo zwraca (patrz Listing 4.5) [Coo00]. W opisywanej aplikacji wzorzec ten zastosowano we wcześniej opisanych klasach:

- *Settings* – potrzebna jest tylko jedna instancja klasy ustawień, dostępna dla każdego elementu aplikacji;
- *ExamPerformance* – w każdym momencie jest możliwość odczytu/zapisu aktualnie badanych pacjentów, przeprowadzanych testów, itp.;
- *XmlManager* – z każdego miejsca aplikacji może następować odczyt/zapis plików XML.

Listing 4.5: Wzorzec Singleton zaimplementowany w klasie *ExamPerformance*.

```
public class ExamPerformance {
    List<AbstractTest> testsToExec = new ArrayList<AbstractTest>();
    List<Series> seriesList = new ArrayList<Series>();
    List<Patient> chosenPatients = new ArrayList<Patient>();
    Patients patientsToSave = new Patients();

    static ExamPerformance instance;
```

```
    public static ExamPerformance getInstance() {
        if(instance == null) {
            instance = new ExamPerformance();
        }
        return instance;
    }
 ...
}
```

4.4 Generyczność

Klasy projektu aplikacji często korzystają z generyczności, która umożliwia użycie sparametryzowanych typów danych. Jeśli dana klasa zawiera pole o typie, który nie jest określony w momencie jej deklaracji lub jest klasą przechowującą elementy o dowolnym typie, definiuje się ją jako klasę generyczną. Dzięki generyczności nie jest konieczne rzutowanie typów podczas dodawania, pobierania lub ustawiania pola/elementu [Fai11].

W opisywanym oprogramowaniu generyczność stosowana jest w kontekście klas abstrakcyjnych i dziedziczenia. Listing 4.6 przedstawia fragment kodu deklarującego klasę abstrakcyjną *AbstractTest* oraz część definicji klasy z niej dziedziczącej.

Listing 4.6: Zastosowanie generyczności w klasach opisujących testy pamięci proceduralnej.

```
public abstract class AbstractTest<C extends
    AbstractTestConfiguration>
{
 protected C myConf ;
 public void setConfiguration (C newConf ) {
  myConf = newConf ;
 }
 ...
 public C getConfiguration () {
  return myConf ;
 }
 public abstract C createConfiguration ();
 ...
}

public class WisconsinTest extends AbstractTest<
    WisconsinConfiguration>
{
```

```
 ...
 @Override
 public WisconsinConfiguration createConfiguration () {
  return new WisconsinConfiguration ();
 }
 ...
}
```

Klasa abstrakcyjna *AbstractTest* zawiera pole o typie, który jest zależny od dziedziczącej z niej klasy, ponieważ inna konfiguracja opisuje każdy z zaimplementowanych testów pamięci proceduralnej. Dzięki generyczności, w momencie tworzenia instancji klasy *WisconsinTest* oraz ustawiania dla niej pola dotyczącego konfiguracji, nie jest konieczne rzutowanie na obiekt klasy *WisconsinConfiguration*. Ponadto, rozwiązanie to zapewnia, że dodanie klasy definiującej i opisującej nowy test jest przejrzyste i łatwe.

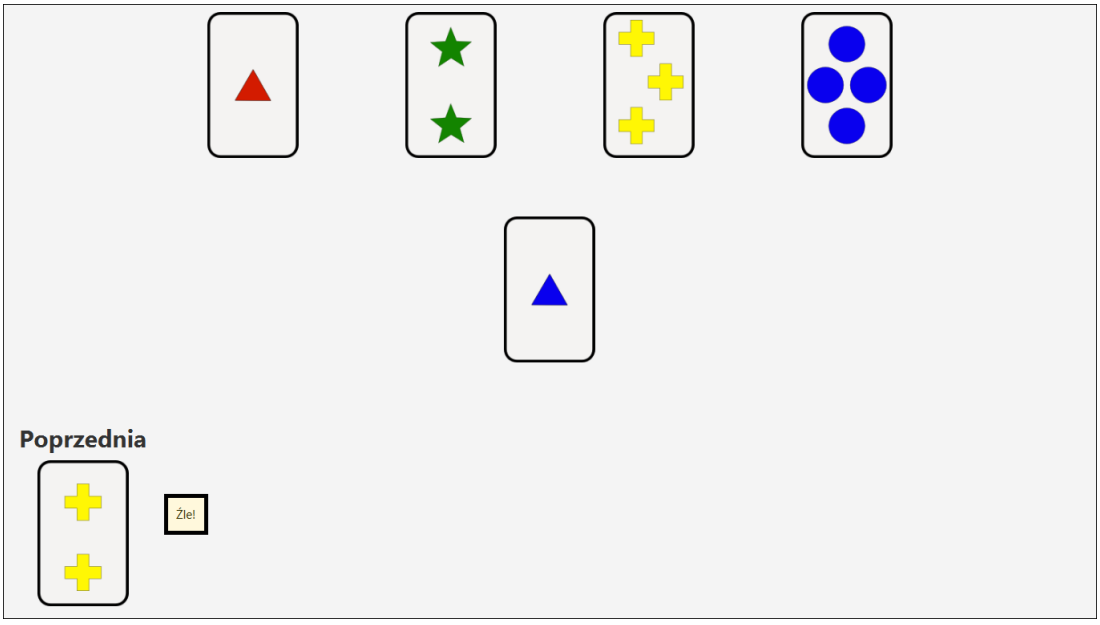
Rozdział 5

Opis implementacyjny oprogramowania

5.1 Moduł testów pamięci proceduralnej

5.1.1 Test sortowania kart z Wisconsin

Ekran testu składa się z elementów stałych – czterech kart wzorcowych i jednej bodźcowej oraz z elementów opcjonalnych, zależnych od wybranej konfiguracji.



Rysunek 5.1: Ekran testu sortowania kart z Wisconsin z wybraną opcją widoczności poprzedniej odpowiedzi wraz z informacją o jej poprawności.

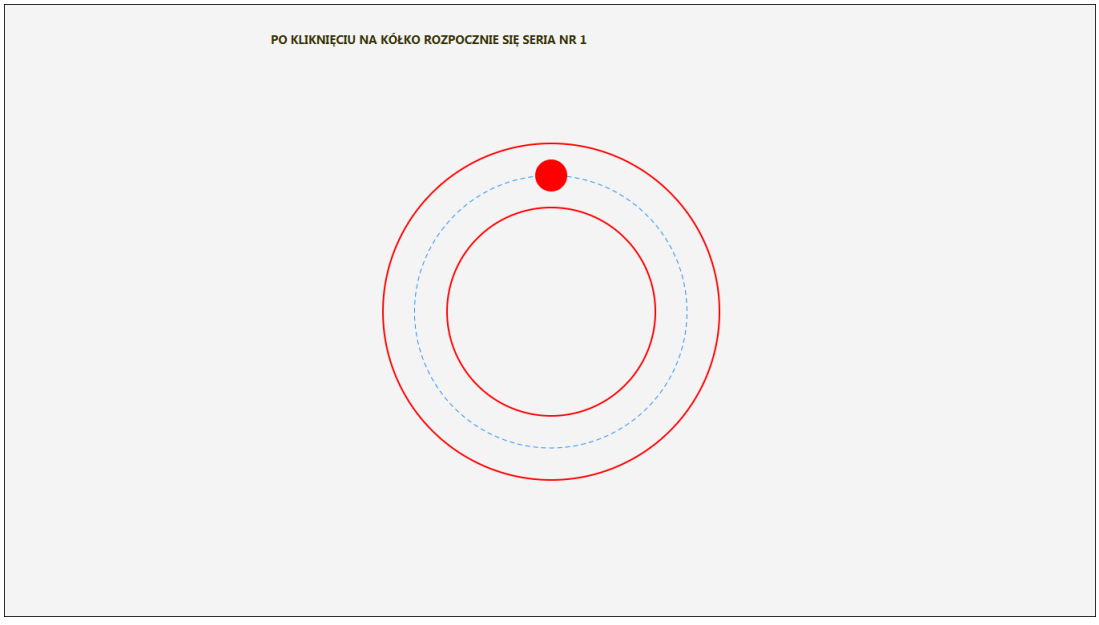
Przed rozpoczęciem wykonania testu tworzona jest lista kolejnych trójek właściwości (kształt, kolor, ilość) kart bodźcowych, które będą prezentowane badanemu. Przy udzieleniu odpowiedzi dokonuje się jej zapis do pliku logu zawierający następujące informacje:

- numer odpowiedzi,
- numer karty wzorcowej, która została wybrana,
- obowiązujące kryterium,
- poprawność odpowiedzi względem obowiązującego kryterium,
- liczba poprawnych odpowiedź z rzędu.

Aktualna kombinacja właściwości zostaje usunięta z listy. Następnie ładowane są kolejne wartości, zgodnie z którymi zmienia się obraz karty. Test kończy się, jeśli lista kombinacji jest pusta lub użytkownikowi udało się zaliczyć sześć kategorii.

5.1.2 Test śledzenia wirnika

Ekran testu składa się z punktu w postaci koła oraz z okrągłej ścieżki po której się on porusza. W momencie inicjalizacji ustawiane są parametry wykonania testu pobierane z przekazanego obiektu konfiguracji.



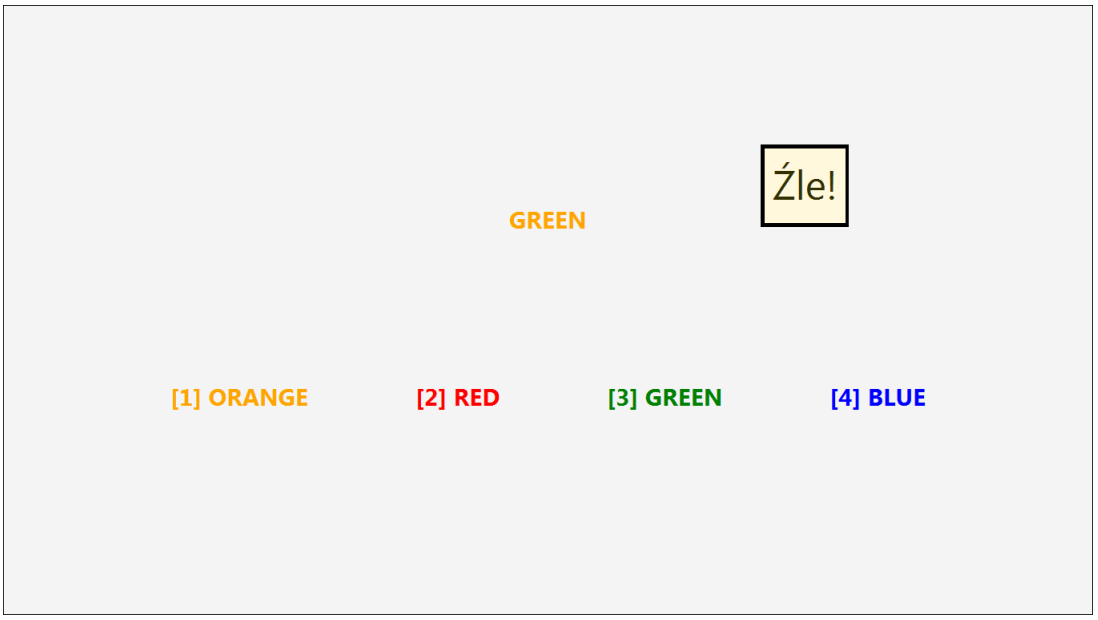
Rysunek 5.2: Ekran startowy testu śledzenia wirnika.

Test rozpoczyna się po kliknięciu w śledzony punkt, co jest wykrywane przez odpowiedni *event handler*. W wyniku tego zdarzenia punkt jest wprawiany w ruch za

pomocą mechanizmu Path transition biblioteki JavaFX. Logowaniu podlegają aktualne współrzędne myszki, współrzędne punktu oraz aktualny czas. Zadanie zapisywania tych danych wykonywane jest przez niezależny wątek wykonywany w ramach mechanizmu Task. Test kończy się po przebiegu określonej w konfiguracji liczby serii.

5.1.3 Test Stroopa

Ekran testu składa się z wyrazu bodźcowego oraz wyrazów oznaczających kolory, zależnych od wybranej konfiguracji.



Rysunek 5.3: Ekran testu Stroopa w wersji nazywania koloru, w którym napisane jest słowo oznaczające inny kolor.

Rozpoczęcie testu następuje po wciśnięciu klawisza 'S'. Ustawiane są pierwsze wartości parametrów wyrazu bodźcowego: kolor i nazwa koloru. Wartości te określają wygląd wyrazu. Przy udzieleniu odpowiedzi następuje zapis do pliku logu następujących informacji:

- numer odpowiedzi,
- wartość wyrazu bodźcowego (kolor lub tekst w zależności od wersji testu),
- wartość odpowiedzi (kolor lub tekst w zależności od wersji testu),
- poprawność,
- numer błędnej odpowiedzi z rzędu,
- czas, który upłynął od rozpoczęcia testu.

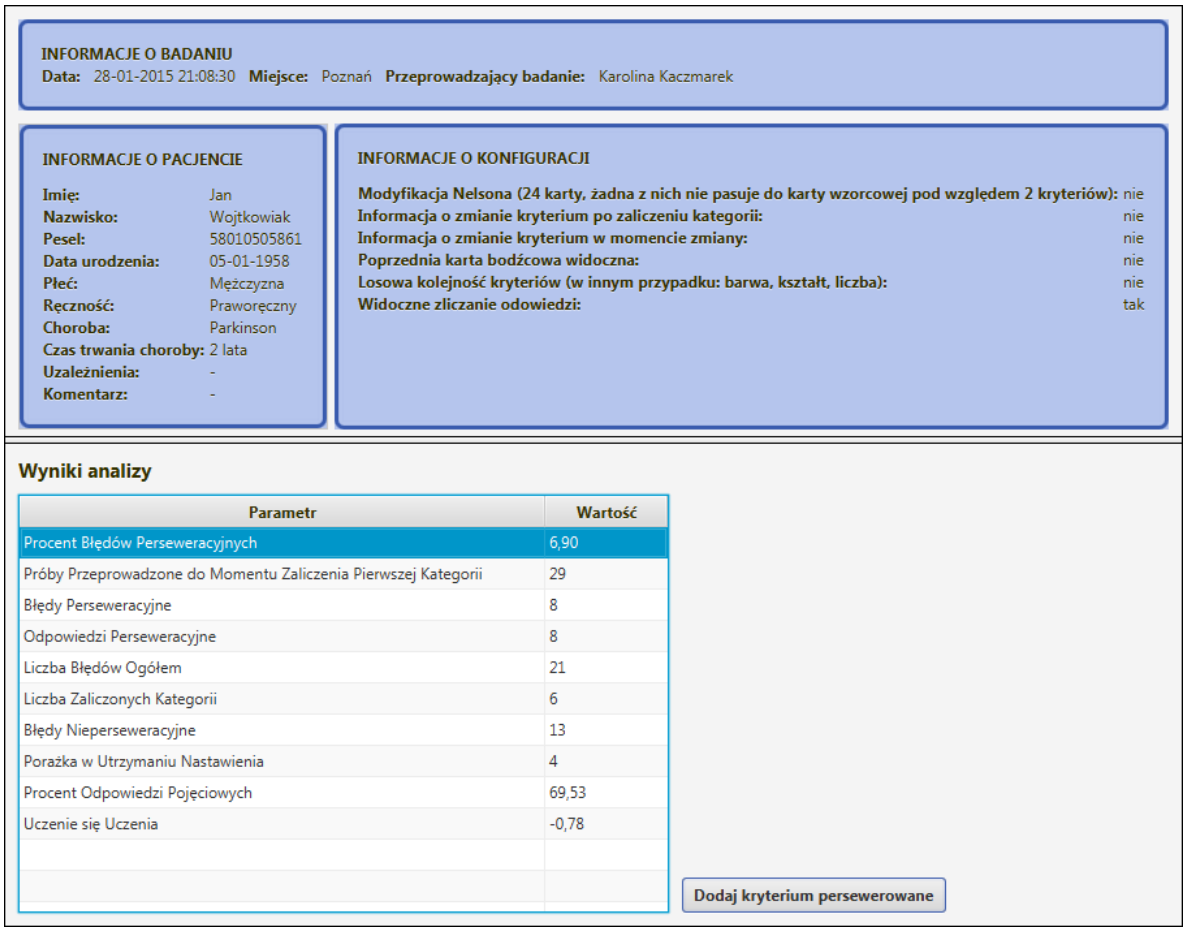
Jeśli odpowiedź jest poprawna następuje zmiana wyrazu bodźcowego. Koniec testu

następuje po udzieleniu zdefiniowanej liczby poprawnych odpowiedzi. Jeśli w konfiguracji wybrana została opcja przeprowadzenia serii próbnej, po udzieleniu pięciu poprawnych odpowiedzi rozpoczyna się właściwy test.

5.2 Moduł analizy

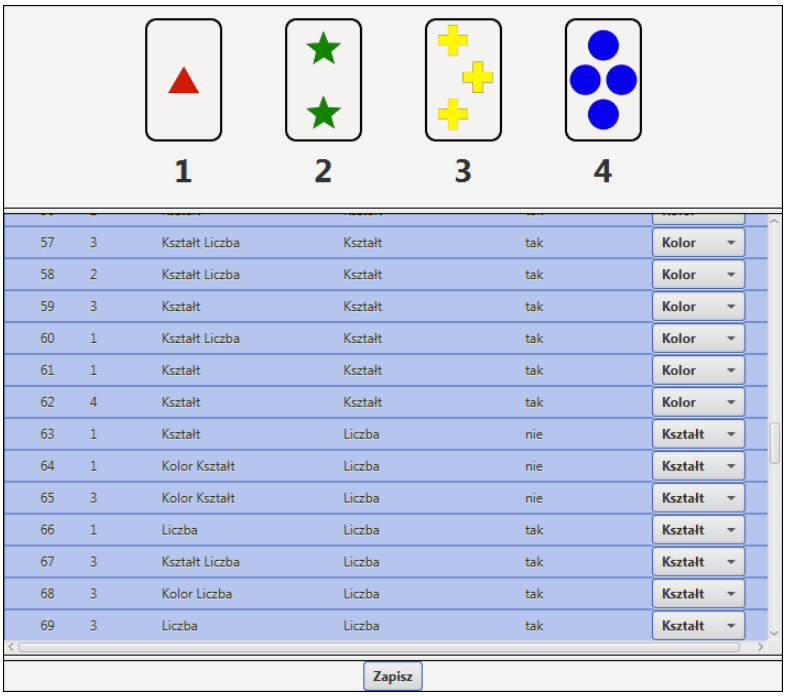
5.2.1 Test sortowania kart z Wisconsin

Analiza testu sortowania kart z Wisconsin opiera się na obliczeniu wartości wskaźników wykonania testu interpretowanych klinicznie. Wyniki przedstawiane są w formie tabelarycznej (patrz Rysunek 5.4).



Rysunek 5.4: Okno analizy pojedynczego testu sortowania kart z Wisconsin.

Część wskaźników wyznaczana jest w oparciu o kryterium persewerowane, które nie jest określane automatycznie. Wskaźniki te liczone są dopiero po wprowadzeniu kryterium do analizowanych badań. Opcja wprowadzenia owego kryterium została udostępniona w oknie analizy jednego testu (patrz Rysunek 5.5).



Rysunek 5.5: Okno wprowadzania kryterium persewerowanego do przebiegu testu.

Wszystkie wskaźniki wykonania testu podlegają analizie statystycznej. Jednak dla wskaźników związanych z kryterium persewerowanym jest ona przeprowadzana tylko, jeśli do każdego z analizowanych badań zostało wprowadzone owe kryterium.

5.2.2 Test śledzenia wirnika

1. Analiza jednego testu

   Na podstawie informacji zapisanych w pliku logu zawierającym przebieg testu obliczanych jest szereg wartości, które zostały zwizualizowane w następujących zakładkach okna analizy:

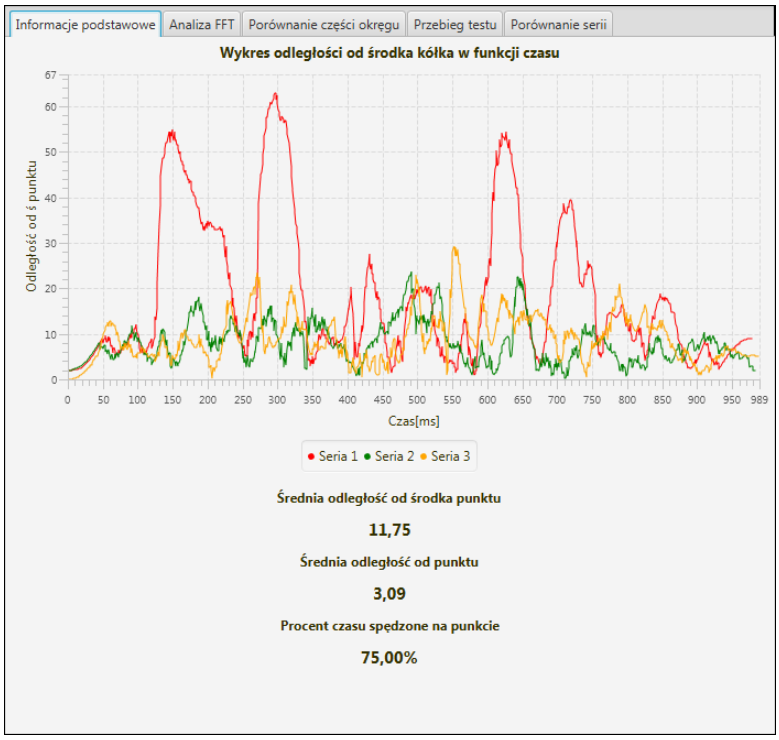
   a) informacje podstawowe – zawiera obliczone podstawowe wartości: średnia odległość od środka oraz od brzegu punktu, procent czasu spędzony na punkcie, a także wykres odległości kursora myszki od środka punktu w funkcji czasu;

   b) analiza FFT – przedstawia wykres analizy FFT dokonanej na podstawie odległości od środka punktu w czasie;

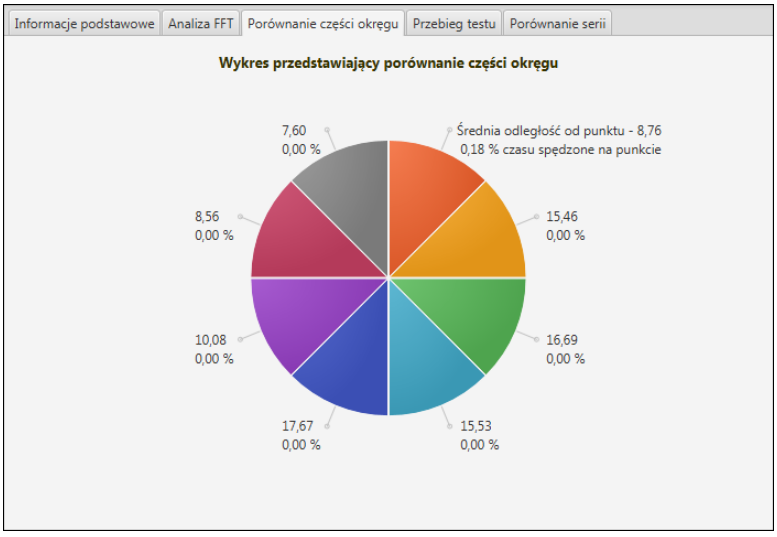
   c) porównania części okręgu – zawiera wykres kołowy (podzielony na osiem fragmentów) pomagający w ocenie, jak w poszczególnych częściach okregu radził sobie badany;

d) przebieg testu – umożliwia odtworzenie przebiegu testu poprzez symulację ruchu punktu i kursora;

e) porównanie serii – zakładka jest dodawana, jeśli przeprowadzono w badaniu więcej niż jedną serię; przedstawia wykresy słupkowe i liniowe porównujące przebieg poszczególnych serii testu.



Rysunek 5.6: Analiza testu śledzenia wirnika – zakładka informacji podstawowych.



Rysunek 5.7: Analiza testu śledzenia wirnika – zakładka porównania części okręgu.

2. Analiza dwóch testów

   Okno tej wersji analizy składa się również z zakładek wymienionych powyżej. W zakładkach przedstawione są jednak wyniki analiz obu badań. Zabieg taki służy pokazaniu różnic w wykonaniu tych testów.

3. Analiza dwóch prób populacji i analiza różnic między populacjami

   Te wersje analizy opierają się na analizie statystycznej (patrz Rozdział 5.2.4). Dla testu śledzenia wirnika analizie statystycznej podlegają następujące parametry:

   - średnia odległość od środka punktu,
   - średnia odległość od brzegu punktu,
   - procent czasu spędzony na punkcie.

5.2.3 Test Stroopa

Analiza testu Stroopa opiera się na obliczeniu następujących parametrów:

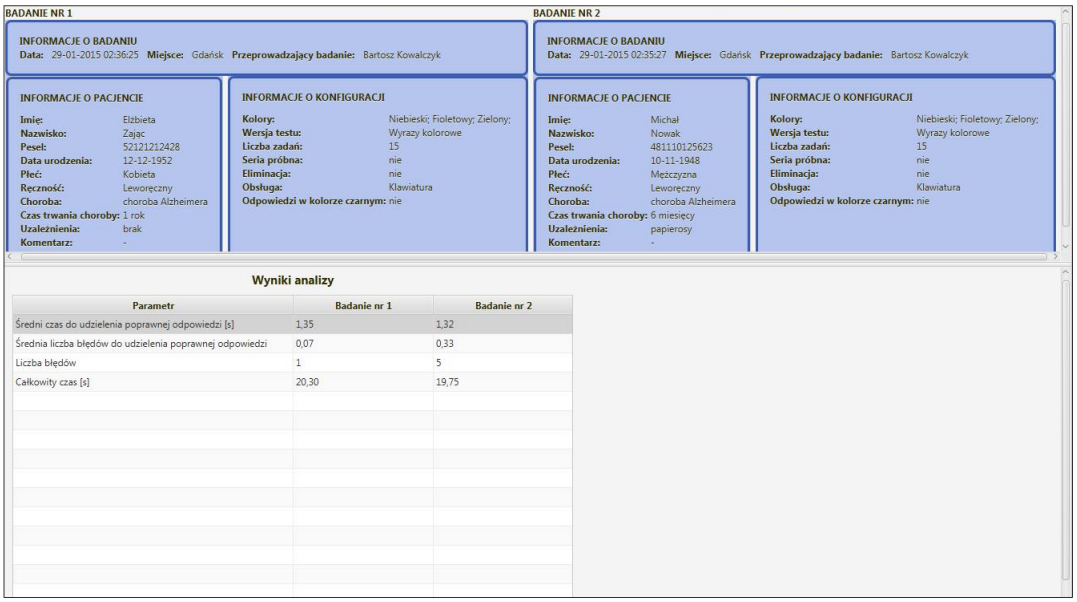
- średni czas potrzebny do udzielenia poprawnej odpowiedzi,
- średnia liczba błędów popełnionych do momentu udzielenia poprawnej odpowiedzi,
- liczba błędów ogółem,
- czas wykonania.

Obliczone wartości, zarówno podczas anlizy jednego, jak i pary testów, przedstawiane są w tabeli (patrz Rysunek 5.8). Analiza statystyczna może być przeprowadzana dla wszystkich wyżej wymienionych parametrów.

5.2.4 Analiza statystyczna

Analiza statystyczna wykorzystywana jest w analizie dwóch prób populacji oraz w analizie różnic między populacjami. Dla każdego z badań należących do prób populacji wyznaczane są parametry podlegające analizie statystycznej, a ich wartości zapisywane są w odpowiednich obiektach klasy rozszerzającej klasę *AbstractPopulationResult*.

Dla każdego z zapisanych parametrów przeprowadzane są testy statystyczne. Rozpoczynają się one od wyboru odpowiedniego testu. Wybór jest podejmowany na podstawie dwóch cech. Pierwszą z nich jest informacja, czy porównywane próby



Rysunek 5.8: Analizy porównawcza pary testów Stroopa.

są zależne. Drugim wyznacznikiem jest informacja, czy rozkłady zmiennych są zbliżone do rozkładu normalnego. Do sprawdzenia drugiej informacji zastosowano test **Kołmogorova-Smirnova** badający podobieństwo rozkładu próby z innym teoretycznym rozkładem – w tym przypadku z rozkładem normalnym. W zależności od wyżej wymienionych cech wybierany i przeprowadzany jest:

- **test t-Studenta dla prób zależnych** – jest to test parametryczny stosowany, jeśli próby są zależne i rozważana cecha ma rozkład normalny;
- **test znakowanych rang Wilcoxona** – test ten stanowi nieparametryczną alternatywę dla testu t-Studenta dla prób zależnych, dlatego stosowany jest w przypadku braku rozkładu normalnego badanej cechy;
- **test t-Studenta dla prób niezależnych** – jest to test parametryczny stosowany, jeśli próby są niezależne i badana cecha ma rozkład normalny;
- **test U Manna Whitneya** – test stanowi nieparametryczną alternatywę dla testu t-Studenta dla prób niezależnych, stosowany, gdy cecha nie ma rozkładu normalnego.

Hipotezy zerowe testów zakładają, że pod względem rozważanej cechy nie występuje istotna statystycznie różnica między populacjami.

W oknie porównania dwóch prób populacji wyświetlana jest tabela zawierająca dla każdego z analizowanych parametrów następujące informacje:

- średnie wartości parametru dla każdej z prób,

- różnicę między tymi średnimi,
- przeprowadzony test statystyczny,
- wyznaczoną p-wartość,
- informację, czy na zadanym poziomie istotności występuje istotna statystycznie różnica między próbami.

Ostatnia informacja uzyskiwana jest przez porównanie obliczonej p-wartości z zadanym poziomem istotności.

W oknie analizy różnic między populacjami wyświetlane są dla obu populacji informacje wymienione powyżej. Taki stan rzeczy służy ułatwieniu porównania różnic między populacjami dla każdego z analizowanych parametrów.

5.3 Eksport danych

Jednym z wymagań funkcjonalnych jest możliwość eksportu badania w taki sposób, aby nie utracić przy tym żadnych informacji o pacjencie, czy o wybranej konfiguracji testu. Klasą odpowiedzialną za logikę eksportu jest *ExportController*, a za interfejs użytkownika ExportWindow. Po wyborze badań do eksportu i przekazaniu ich z warstwy widoku do warstwy kontrolera tworzony jest tymczasowy folder o nazwie *nazwa_pliku_z_przebiegiem_testu_export*, do którego kopiowany jest zapisany wcześniej przebieg testu. Następnie w katalogu tworzone są pliki XML:

- *patient.xml* – zapisane są tutaj dane pacjenta, któremu przeprowadzono eksportowane badanie,
- *exam.xml* – zawiera informacje o badaniu: data, miejsce, badający, konfiguracja testu.

W kolejnym kroku tymczasowy folder jest kompresowany do pliku .zip. Powstałe archiwum zostaje umieszczone w podkatalogu *exported* katalogu roboczego.

Eksport całej serii badań jest realizowany w sposób analogiczny. Tworzony jest folder o nazwie *nazwa_serii_series_export*, a w nim plik *series.xml* z zapisanym obiektem serii badań. Następnie, podobnie jak w przypadku jednego testu, eksportowane i zapisywane są badania wchodzące w skład serii. W ostatnim kroku procesu cały katalog tymczasowy jest kompresowany. Kompresja folderów dokonywana jest za pomocą klasy z pakietu java.util.zip *ZipOutputStream*.

5.4 Import danych

Umożliwienie importu danych o realizacji testu wraz z niezbędnymi informacjami o pacjencie czy szczegółach badania stanowi kolejne wymaganie zdefiniowane dla systemu. Ta funkcjonalności ułatwia korzystanie z platformy oraz współpracę wielu użytkowników.

Klasą odpowiedzialną za logikę importu jest *ImportController*, a odpowiadającą za warstwę widoku *ImportWindow*. Po wskazaniu przez użytkownika pliku/plików wyeksportowanych z platformy, pierwszym krokiem jest dekompresja pliku .zip z badaniem, a następnie walidacja zawartości powstałego folderu. Jeśli walidacja przebiegła pomyślnie, sprawdza się, czy w katalogu znajduje się plik opisujący serię – w takim przypadku importowana jest seria badań, w przeciwnym razie pojedyncze badanie.

Import pojedynczego badania polega na skopiowaniu importowanego pliku przebiegu testu do katalogu *tests* zawierającego przebiegi przeprowadzanych testów. Kolejnym krokiem jest pobranie z pliku *patient.xml* pacjenta i sprawdzenie, czy występuje on wśród lokalnie zapisanych. Jeśli tak, to do jego listy przeprowadzonych badań dodawane jest zaimportowane badanie (z pliku *exam.xml*). W przeciwnym wypadku, pacjent jest dopisywany do lokalnie zapisanych pacjentów.

W przypadku importu serii z pliku importowanego *series.xml* pobierany jest obiekt serii, który dopisywany jest do lokalnie zapamiętanych serii. Następnie przeprowadzany jest wyżej opisany import pojedynczych badań tej serii.

5.5 Instrukcja dodania nowego testu

Jednym z najważniejszych wymagań zdefiniowanych dla platformy było zaprojektowanie jej w taki sposób, aby istniała możliwość poszerzania jej funkcjonalności poprzez dodawanie kolejnych implementacji, na przykład Testu Rysowania w Odbiciu Lustrzanym (ang. Mirror Tracing Task) czy Testu Predykcji Pogody (ang. Weather Prediction Task). W dalszej części rozdziału przedstawiono instrukcję opisującą poprawne wprowadzenie nowego testu do platformy.

5.5.1 Przebieg testu

W celu dodania do platformy implementacji nowego testu należy wykonać następujące kroki:

1. Stworzenie pakietu, który będzie zawierał klasy dotyczące nowego testu w pakiecie testModule.

2. Stworzenie odpowiednich plików dla nowego testu:
    - w pakiecie testModule/configuration/view klasy konfiguracji dziedziczącej z klasy *AbstractTestConfiguration*,
    - w pakiecie testModule/configuration/controller klasy dziedziczącej z klasy *AbstractConfigurationController*,
    - w folderze *fxml* dokumentu FXML przedstawiającego zobrazowane parametry konfiguracji.
3. Stworzenie klasy dziedziczącej z klasy *AbstractTest* z parametrem generycznym w postaci klasy konfiguracji. Konieczne jest dopisanie adnotacji @XmlRootElement przed definicją nowej klasy.
4. Dodanie w adnotacji @XmlSeeAlso w klasie *AbstractTest* nowej klasy w postaci *NowyTest.class*
5. Dodanie nowego obiektu do listy dostępnych testów w klasie *TestFactory*.

5.5.2 Analiza testu

Stworzenie klas odpowiedzialnych za analizę wyników dodanego testu wymaga postępwania zgodnie z poniższą instrukcją:

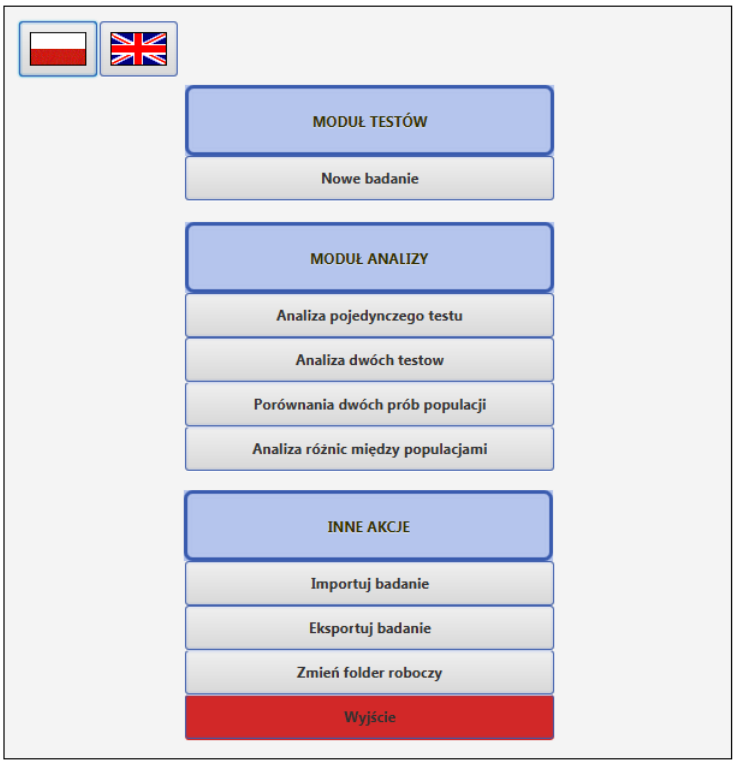
1. Stworzenie nowego pakietu w pakiecie psychomotorTestAnalysis.
2. Dodanie klas dziedziczących z *AbstractResult* i *AbstractPopulationResult* w nowym pakiecie.
3. W celu wprowadzenia wszystkich rodzajów analizy stworzenie klas dziedziczących z:
    - *AbstractOneTestAnalysis*,
    - *AbstractTwoTestsAnalysis*,
    - *AbstractTwoTriesPopulationAnalysis*,
    - *AbstractTwoPopulationsCompareAnalysis*.

Rozdział 6

Instrukcja korzystania z aplikacji

W tym rozdziale opisano działanie systemu z perspektywy użytkownika.

Po uruchomieniu aplikacji użytkownik zostaje poproszony o wybranie folderu roboczego, w którym zapisywane będą przebiegi testów, informacje o pacjentach oraz dane wyeksportowane z systemu. Następnie zostaje wyświetlone główne okno platformy.

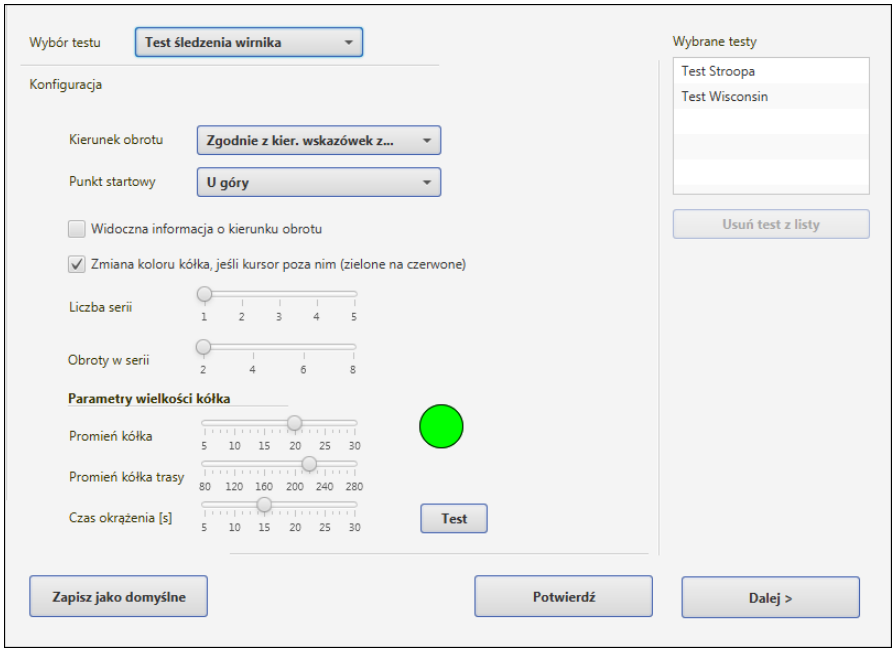


Rysunek 6.1: Okno główne platformy.

Okno główne udostępnia możliwość przełączenia języka oraz zawiera menu główne składające się z trzech części: modułu testów, modułu analizy oraz innych opcji.

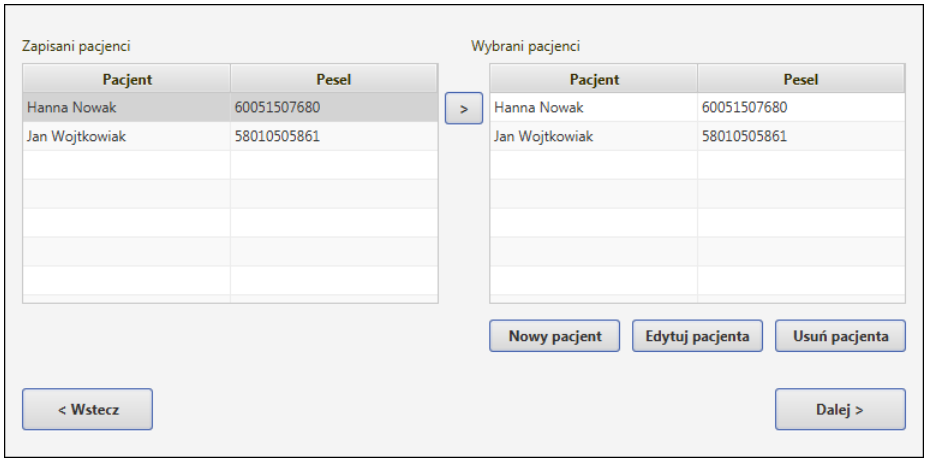
6.1 Moduł testów

Tworzenie nowego badania zostało podzielone na kilka etapów. W pierwszym z nich użytkownik proszony jest o wybór testów, które chciałby przeprowadzić. Szczegóły dotyczące parametryzacji poszczególnych testów przedstawiono w Rozdziale 3.3.



Rysunek 6.2: Okno wyboru testów do przeprowadzenia w ramach badania.

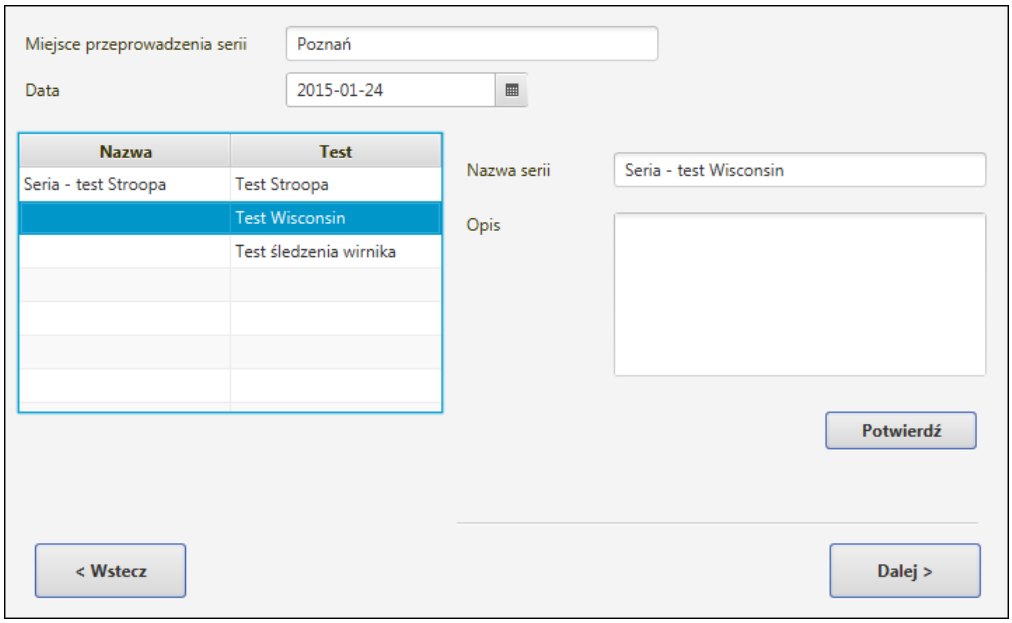
Następnym krokiem jest wybór pacjentów. Można wybrać ich spośród wcześniej badanych osób lub wprowadzić nowe dane. Dodatkowo istnieje możliwość edycji informacji o zapisanych pacjentach.



Rysunek 6.3: Okno wyboru pacjentów, na których ma zostać przeprowadzone badanie.

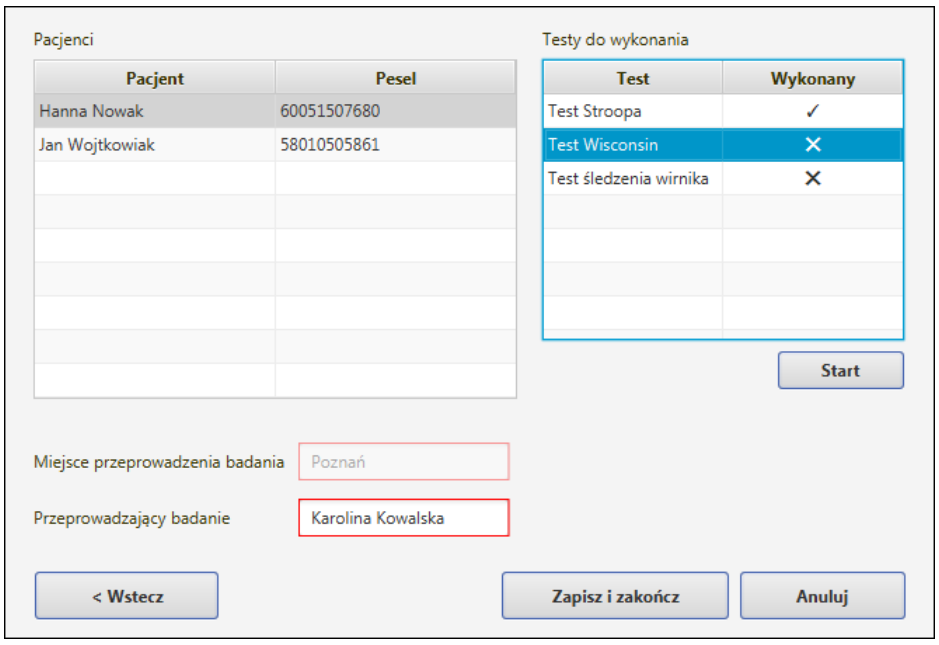
Po wybraniu więcej niż jednego pacjenta użytkownik otrzymuje informację o moż-

liwości utworzenia serii, która w znaczący sposób ułatwia późniejszy proces wyboru badań do analizy prób populacji.



Rysunek 6.4: Okno zapisu serii badań.

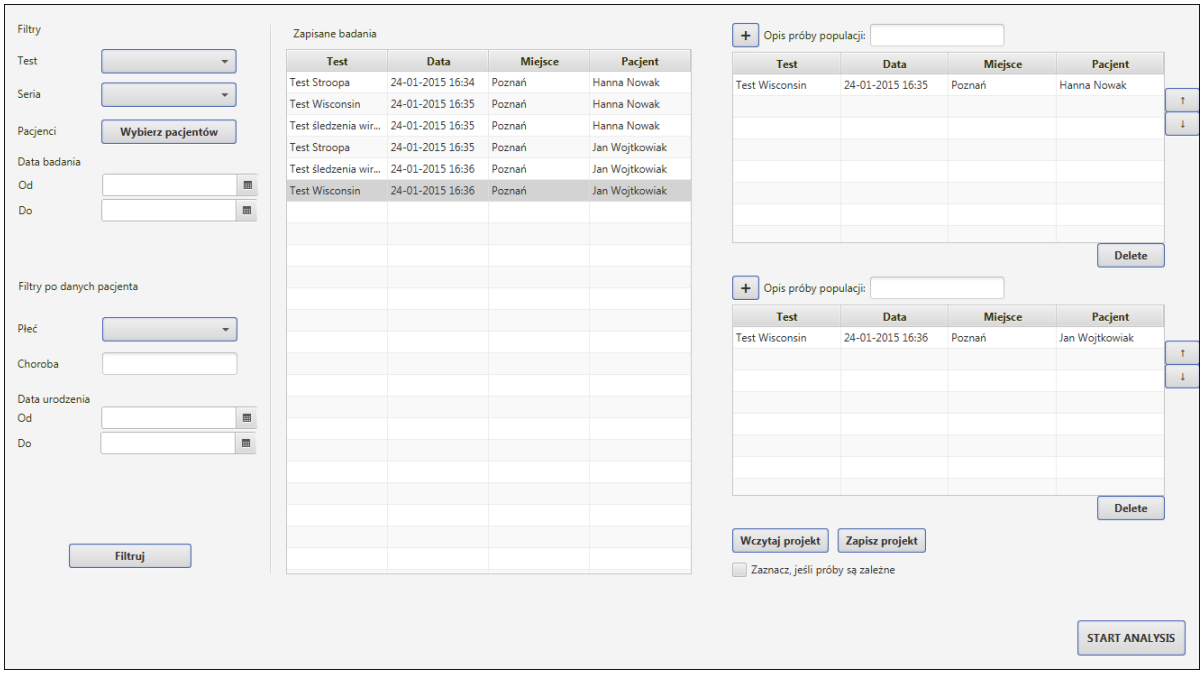
Kolejny ekran systemu podsumowuje wcześniejsze etapy tworzenia nowego badania i pozwala na zarządzanie procesem wykonywania kolejnych testów.



Rysunek 6.5: Ekran zarządzania procesem wykonywania testów.

6.2 Moduł analizy

Moduł analizy udostępnia możliwość przeprowadzenia czterech rodzajów analizy wyników badań. Ekran wyboru testów do każdej z wersji zawiera tabelę prezentującą zapisane badania oraz panel filtrów umożliwiających selekcję badań.



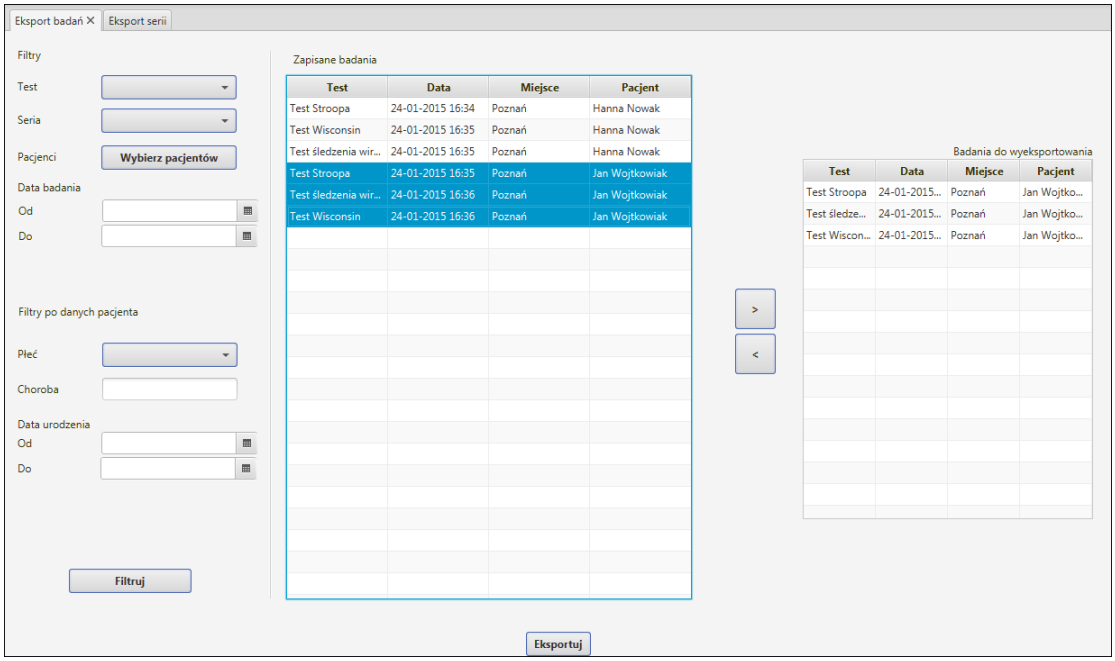
Rysunek 6.6: Okno wyboru badań do analizy prób populacji.

W przypadku analizy dwóch prób populacji użytkownik musi dodać interesujące go pozycje tabeli badań do tabel reprezentujących próby oraz dodatkowo określić, czy zdefiniowane próby są zależne.

6.3 Inne opcje

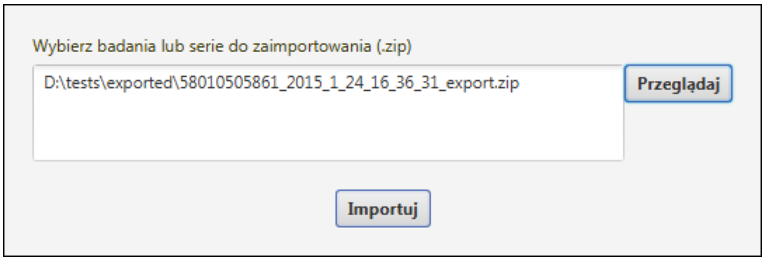
Na moduł innych opcji składają się: możliwość eksportu i importu badań oraz zmiany folderu roboczego.

Platforma umożliwia eksport zarówno pojedynczych badań, jak i ich serii. Użytkownik musi wybrać z tabeli interesujące go pozycje. Po wyeksportowaniu badań archiwa zapisywane są w podkatalogu *exported* folderu roboczego.



Rysunek 6.7: Ekran wyboru badań do wyeksportowania.

W celu zaimportowania badań do systemu użytkownik powinien wskazać, które badania chciałby zaimportować, wybierając odpowiednie archiwa ZIP.



Rysunek 6.8: Ekran importu badań.

Rozdział 7

Podsumowanie

Efektem pracy jest platforma do realizacji i analizy testów pamięci proceduralnej. Zrealizowane zostały wszystkie postawione wymagania. Aplikacja umożliwia przeprowadzenie trzech testów (testu sortowania kart z Wisconsin, testu śledzenia wirnika oraz testu Stroopa). Każdy z nich został rozszerzony pod względem możliwości konfiguracyjnych w porównaniu do ich pierwotnych wersji. Zapis badań jest bardzo dokładny, co pozwala na precyzyjne odtworzenie ich przebiegów. Ponadto platforma umożliwia analizę wyników testów, wyznaczając przy tym więcej parametrów oceniających zdolności pacjenta w porównaniu z innymi dostępnymi aplikacjami. Dodatkowa analiza z wykorzystaniem testów statystycznych pomaga w ocenie istotności zmian zachodzących w badanych próbach pacjentów.

Poza podstawowymi możliwościami zrealizowane zostały dodatkowe funkcje ułatwiające pracę z platformą. Importowanie i eksportowanie badań sprzyja współpracy lekarzy, a możliwość wyboru języka aplikacji poszerza grono odbiorców. Aplikacja zaimplementowana została z uwzględnieniem możliwości jej późniejszego rozwoju. Modularność oraz zastosowanie odpowiednich wzorców projektowych ułatwia dodanie kolejnych testów.

Platforma został stworzona z myślą o przyszłych jej użytkownikach, dlatego duży nacisk położono na uzyskanie komfortu obsługi aplikacji, w tym przede wszystkim zaprojektowanie intuicyjnego w obsłudze interfejsu graficznego. Dokładnie przemyślano każdy krok prowadzący do wykonania badania, a wyniki analizy starano się przedstawić w przejrzysty sposób.

Największą trudnością napotkaną w czasie realizacji pracy okazało się zaplanowanie i zaprojektowanie systemu w taki sposób, aby uwzględnić w niej testy o bardzo różnym charakterze. Kolejnym problemem był brak relacyjnej bazy danych, co wymusiło potrzebę ustalenia sposobu ewidencji pacjentów i badań w pliku XML. Znalezienie dobrych rozwiązań znacznie ułatwiło pracę nad poszczególnymi częściami

systemu.

Realizacja projektu zdecydowanie podniosła kwalifikacje członków zespołu, którzy zapoznali się z nowymi technologiami oraz poszerzyli wiedzę dotyczącą znanych już narzędzi. Konieczność dokładnego zaplanowania działania niektórych części systemu sprawiła, że grupa nie tylko nabyła większe doświadczenie ze strony programowania, ale również zyskała swobodę w projektowaniu aplikacji. Znajomość technologii JavaFX z pewnością pomoże w przyszłości w efektywnym tworzeniu interfejsu graficznego użytkownika, a przetwarzanie danych umieszczonych na zewnątrz aplikacji stanie się sprawniejsze dzięki poznaniu technolgoii JAXB. Narzędzia Apache Maven oraz Git zaś są używane przez profiesjonalne zespoły programistyczne, stąd umiejętność pracy z nimi jest istotna z perspektywy realizacji innych systemów. Napotkane problemy i starania w znalezieniu rozwiązań również rozszerzyły kompetencje programistów.

Praca nad projektem rozwinęła zdolność komunikacji w zespole. Rozszerzyła także umiejętność właściwego podziału zadań, a także poczucie odpowiedzialności za wykonywaną część platformy. Doświadczenie zdobyte podczas wspólnego tworzenia oprogramowania jest niezwykle cenne i kluczowe dla realizacji projektów w większych grupach programistycznych.

Bibliografia

[AAF13] David B. Arciniegas, C. Alan Anderson, and Christopher M. Filley. *Behavioral Neurology and Neuropsychiatry.* Cambridge University Press, 2013.

[Ada52] Jack A. Adams. *Warm-up decrement in performance on the pursuit-rotor.* The American Journal of Psychology, 1952.

[Amm55] Robert B. Ammons. *Rotary pursuit apparatus: I. Survey of variables.* Psychological Bulletin, 1955.

[Coo00] James William Cooper. *Java Design Patterns: A Tutorial.* Addison-Wesley Professional, 2000.

[Fai11] Yakov Fain. *Java Programming 24-Hour Trainer.* John Wiley and Sons, 2011.

[Gre08] Marek Grebski. *Sukces na egzaminie.* WSiP, 2008.

[Jaw02] Aleksandra Jaworska. *Test Sortowania Kart z Wisconsin. Podręcznik.* Pracownia Testów psychologicznych PTP, 2002.

[MAV] http://4programmers.net/java/maven.

[PEB] http://pebl.sourceforge.net.

[Som11] Ian Sommerville. *Software Engineering 9th Edition.* Pearson, 2011.

[SSS06] Otfried Spreen, Spreen Strauss, and Elisabeth Sherman. *A Compendium of Neuropsychological Tests: Administration, Norms, and Commentary.* Oxford University Press, 2006.

[Ste92] Michał Steuden. *Test Sortowania Kart Wisconsin (Wisconsin Card Sorting Test-WCST) -interpretacja kliniczna s.73-96 w Wybrane zagadnienia z psychologii klinicznej.* Norbertinum, 1992.

[Str35] John Ridley Stroop. *Studies of interference in serial verbal reactions.* Journal of Experimental Psychology, 1935.