PROJEKT: Aplikacja do zarządzania samochodami wspólnoty mieszkaniowej
Autorki: Hanna Laszkiewicz, Kalina Białek

1. TESTOWANIE WYMAGAŃ FUNKCJONALNYCH

Warunek testowy nr 1

lp.	nazwa	opis	oczekiwany rezultat
1	Logowanie	Sprawdzenie funkcji logowania do systemu.	Można się zalogować do systemu. System jest bezpieczny i nie pozwala na zalogowanie się nieuprawnionych osób. Można się zalogować jako administrator lub użytkownik.

Przypadki testowe

lp.	priorytet	nazwa	warunki wstępne	kroki wykonania	oczekiwany rezultat
1	1	Próba zalogowania po wpisaniu poprawnego loginu i hasła.	W bazie systemu znajduje się użytkownik/ administrator, na którego konto można spróbować się zalogować.	1. Wpisanie poprawnego loginu oraz hasła. 2. Wciśnięcie przycisku “zaloguj się”.	Użytkownik/ administrator został zalogowany do systemu. System przechodzi do ekranu startowego.
2	2	Próba zalogowania po wpisaniu poprawnego loginu i niepoprawnego hasła	W bazie systemu znajduje się użytkownik/ administrator, na którego konto można spróbować się zalogować.	1. Wpisanie poprawnego loginu oraz niepoprawnego hasła. 2. Wciśnięcie przycisku “zaloguj się”.	Użytkownik/ administrator nie został zalogowany do systemu. System wyświetla komunikat “niepoprawne hasło”. System pozostaje na ekranie logowania.
3	2	Próba zalogowania po wpisaniu niepoprawnego loginu i niepoprawnego hasła.	-	1. Wpisanie niepoprawnego loginu oraz niepoprawnego hasła. 2. Wciśnięcie przycisku “zaloguj się”.	Użytkownik/ administrator nie został zalogowany do systemu. System wyświetla komunikat “nie istnieje użytkownik o takim loginie”. System pozostaje na ekranie logowania.

Warunek testowy nr 2

lp.	nazwa	opis	oczekiwany rezultat
1	Rezerwacja samochodu	Sprawdzenie funkcji rezerwacji samochodu na określony termin przez użytkownika.	Można zarezerwować samochód. System nie pozwala na rezerwację tego samego samochodu w nakładające się terminy. System poprawnie wprowadza do bazy dokonaną rezerwację.

Przypadki testowe

lp.	priorytet	nazwa	warunki wstępne	kroki wykonania	oczekiwany rezultat
1	1	Próba rezerwacji samochodu dostępnego w podanym terminie.	W bazie systemu znajduje się użytkownik i może dokonać rezerwacji. W bazie systemu znajduje się samochód, który jest udostępniony do wypożyczenia. Istnieje pusty termin, na który można zarezerwować samochód.	1. Zalogowanie się. 2. Wciśnięcie przycisku “Rezerwuj samochód”. 3. Wprowadzenie poprawnego terminu rozpoczęcia rezerwacji. 4. Wciśnięcie przycisku “zatwierdź”. 5. Wprowadzenie poprawnego terminu zakończenia rezerwacji. 6. Wciśnięcie przycisku “zatwierdź”. 7. Wybranie samochodu do rezerwacji. 8. Potwierdzenie rezerwacji.	Rezerwacja została dokonana. Rezerwacja została poprawnie dodana do bazy systemu. System przechodzi do ekranu startowego.
2	2	Próba rezerwacji samochodu w terminie, kiedy nie ma dostępnego żadnego samochodu.	W bazie systemu znajdują się użytkownicy i mogą dokonać rezerwacji. W bazie systemu znajduje się samochód, który jest	1. Zalogowanie się. 2. Wciśnięcie przycisku “Rezerwuj samochód”. 3. Wprowadzenie poprawnego terminu rozpoczęcia rezerwacji.	System wyświetla komunikat “brak dostępnych samochodów w podanym terminie”. Rezerwacja nie została dokonana. System pozostaje na ekranie wyboru terminu.

			udostępniony do wypożyczenia. W bazie systemu znajduje się rezerwacja.	4. Wciśnięcie przycisku “zatwierdź”. 5. Wprowadzenie poprawnego terminu zakończenia rezerwacji. 6. Wciśnięcie przycisku “zatwierdź”.	
3		Próba rezerwacji samochodu na niepoprawnie podany termin.	-	1. Zalogowanie się. 2. Wciśnięcie przycisku “Rezerwuj samochód”. 3. Wprowadzenie niepoprawnego terminu rozpoczęcia rezerwacji. 4. Wciśnięcie przycisku “zatwierdź”. 5. Wprowadzenie niepoprawnego terminu zakończenia rezerwacji. 6. Wciśnięcie przycisku “zatwierdź”.	System wyświetla komunikat “niepoprawny termin rezerwacji”. Rezerwacja nie została dokonana. System pozostaje na ekranie wyboru terminu.

Warunek testowy nr 3

lp.	nazwa	opis	oczekiwany rezultat
1	Dodawanie samochodu	Sprawdzenie funkcji dodania samochodu do bazy przez administratora.	Można dodać samochód do bazy. System nie pozwala dwukrotnie dodać tego samego samochodu do bazy. System poprawnie dodaje samochód do bazy.

Przypadki testowe

lp.	priorytet	nazwa	warunki wstępne	kroki wykonania	oczekiwany rezultat
1	1	Próba dodania pierwszy raz samochodu i podanie poprawnych danych.	W bazie znajduje się administrator i może dodać samochód.	1. Zalogowanie się. 2. Wciśnięcie przycisku “Dodaj samochód”. 3. Wprowadzenie poprawnych danych samochodu. 4. Wciśnięcie przycisku “zatwierdź”.	Samochód został dodany do bazy. System przechodzi do ekranu startowego.
2	2	Próba dodania samochodu, który już jest w bazie.	W bazie znajduje się administrator i może dodać samochód. W bazie znajduje się samochód.	1. Zalogowanie się. 2. Wciśnięcie przycisku “Dodaj samochód”. 3. Wprowadzenie poprawnych danych samochodu, który już jest w bazie. 4. Wciśnięcie przycisku “zatwierdź”.	System wyświetla komunikat “samochód już znajduje się w bazie”. Samochód nie został dodany do bazy. System pozostaje na ekranie dodawania samochodu.

2. TESTOWANIE WYMAGAŃ NIEFUNKCJONALNYCH

Kryterium	Opis	Podstawa w dokumentacji	Jak to sprawdzimy?
Obsługa aplikacji zrozumiała dla użytkownika	Obliczymy, jaką część interfejsu użytkownik może dostosować, a testerzy sprawdzą, czy komunikaty wyświetlane w aplikacji są zrozumiałe. Ponadto zapewnimy dostęp do instrukcji obsługi aplikacji i opisu najważniejszych funkcji oraz dostarczymy odpowiednią dokumentację.	N5556_ISO: User interface aesthetics, Message clarity, Completeness of user documentation and/or help facility	Obliczymy następujący iloraz: $\frac{liczba\ \ zrozumiałych\ komunikatów}{liczba\ wszystkich\ komunikatów} * 100\%$ Chcemy osiągnąć 100% zrozumiałych komunikatów. Sprawdzimy, czy dostosowywanie wyglądu aplikacji obejmuje m. in. możliwość zmiany stylu i wielkości czcionki. Powinna także być możliwość dostosowania powiadomień. Przeanalizujemy dostępność zrozumiałej dla użytkownika instrukcji obsługi (np. samouczek przy pierwszym logowaniu do aplikacji i po każdej aktualizacji - dla nowych funkcji) oraz kontaktu z obsługą w przypadku pytań i sugestii.
Łatwość wprowadzania danych przez administratora	Administrator powinien łatwo dowiedzieć się z dokumentacji, jak wprowadzić dane o samochodach i wspólnocie.	N5556_ISO: Learnability	Obliczymy następujący iloraz: $\frac{liczba\ funkcji\ z\ instrukcją\ do\ nich}{liczba\ wszystkich\ funkcji} * 100\%$ Dopilnujemy, aby dokumentacja zawierała opisy wszystkich funkcji używanych do wprowadzania danych na temat wspólnoty, tak, aby osiągnięty wynik wynosił 100%.
Łatwość konserwacji	Aby łatwiej było wprowadzać zmiany, aktualizować aplikację i naprawiać błędy, chcemy, aby nasza aplikacja składała się z modułów/komponentów.	N5556_ISO: Modularity, Condensability	Obliczymy następujący iloraz: $\frac{liczba\ komponentów\ niezależnych}{liczba\ wszystkich\ komponentów} * 100\%$ Poziom dopuszczalny: 60%, Poziom satysfakcjonujący: 70%, Poziom optymalny: 80%. Komponenty niezależne to takie, na które nie ma wpływu zmiana innych modułów.

Wielojęzyczność	Planujemy dostępność co najmniej w językach polskim i angielskim.	brak	Z pomocą tłumacza zatwierdzimy poprawność tłumaczenia aplikacji na język angielski.
Projekt w myśl zasady Accessibility	Sprawdzimy z jakiej części funkcji może skorzystać osoba z niepełnosprawnością.	N5556_ISO: Accessibility	Obliczymy następujący iloraz: $\frac{liczba\ funkcji\ \ dostępnych\ \ dla\ OzN}{liczba\ wszystkich\ funkcji} * 100\%$ Celem będzie 100%, aby aplikacja była jak najbardziej dostępna dla OzN.

3. PLAN BETA TESTÓW

a. Cel testowania:
   i. zbadanie funkcjonalności, wydajności i stabilności aplikacji do zarządzania samochodami wspólnoty mieszkaniowej
   ii. znalezienie ewentualnych problemów
   iii. uzyskanie opinii potencjalnych użytkowników
b. Etapy testowania:
   i. przygotowanie do testów:
      - wybór testerów - należy zapewnić reprezentatywną grupę testerów, tzn. różnorodną względem np. umiejętności technicznych i wieku
      - ustalenie sposobu komunikacji z testerami
      - utworzenie odpowiedniego środowiska testowego, pozwalającego na testowanie różnych scenariuszy użycia
   ii. planowanie testów:
      - ustalenie kryteriów sukcesu dla poszczególnych funkcji
      - opracowanie szczegółowych przypadków testowych z uwzględnieniem różnych systemów operacyjnych i urządzeń
      - przygotowanie narzędzia do wyłapywania błędów i analizy wydajności aplikacji
   iii. wykonanie testów:
      - testy funkcjonalne - sprawdzenie poprawności poszczególnych funkcji
      - testy kompatybilności - sprawdzenie poprawności działania aplikacji w zależności od użytego systemu operacyjnego i urządzenia
      - testy wydajności - sprawdzenie szybkości działania aplikacji na różnych urządzeniach i systemach, m. in. czasu potrzebnego na odpowiedź z bazy danych, ładowanie poszczególnych ekranów aplikacji
      - testy bezpieczeństwa - sprawdzenie zgodności z prawem dotyczącym danych osobowych oraz zabezpieczenia bazy danych
   iv. zbieranie informacji i uwag od testerów:
      - komunikacja na bieżąco przy użyciu wybranego wcześniej kanału
   v. analiza wyników:
      - zapoznanie się z opiniami, sugestiami
      - naprawianie usterek na bieżąco
      - sporządzenie analizy częstotliwości błędów, nadanie im priorytetów rozwiązania
   vi. powtórzenie testów
      - po rozwiązaniu wszystkich zgłoszonych problemów, w celu sprawdzenia skuteczności napraw
   vii. przygotowanie do wdrożenia oprogramowania: zakończenie testów i sporządzenie raportu, zaktualizowanie dokumentacji
c. harmonogram testów:

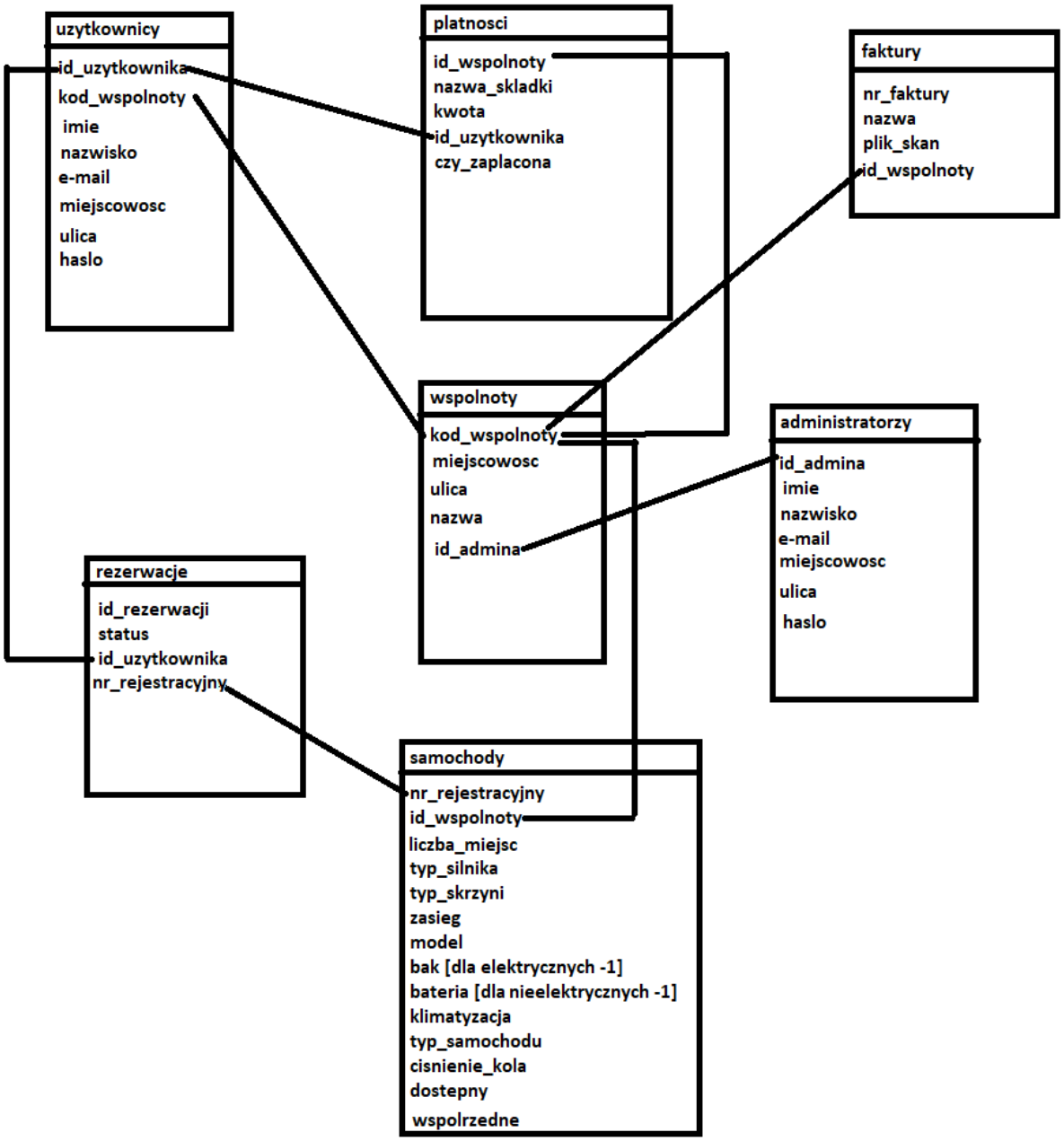
    i. daty początkowa i końcowa - do ustalenia
    ii. wdrożenie aplikacji bezpośrednio po zakończeniu beta testów i naprawiania błędów
d. kryteria sukcesu beta testów:
    i. poprawienie wszystkich błędów zgłoszonych przez testerów
    ii. co najmniej 90% opinii pozytywnych

4. PLAN ZARZĄDZANIA JAKOŚCIĄ OPROGRAMOWANIA

a. Celem planu zarządzania jakością naszego oprogramowania jest zapewnienia wysokiej jakości oprogramowania oraz sprawne zarządzanie jej rozwojem i utrzymaniem.
b. Zespół zarządzający procesem zapewnienia wysokiej jakości oprogramowania:
    i. Quality Assurance Manager - osoba odpowiedzialna za zespół
    ii. testerzy i analitycy - odpowiedzialni za przeprowadzenie testów i sporządzanie raportów
c. Narzędzia, technologie i metodologie utrzymania jakości
    i. zebranie wymagań funkcjonalnych i niefunkcjonalnych oraz zapewnienie zgodności oprogramowania z nimi
    ii. testowanie kodu na etapie tworzenia, kontrola działania kodu na bieżąco
    iii. dokładne testowanie nowych funkcji
    iv. stały kontakt z użytkownikami aplikacji i wdrażanie sugestii oraz naprawianie błędów na bieżąco
    v. testowanie beta
    vi. przejścia (walkthrough) w celu utrzymania czytelności kodu oraz dokumentów, by ułatwić utrzymywanie aplikacji po wypuszczeniu
d. Testowanie oprogramowania
    i. testy jednostkowe - aby sprawdzić poprawne działanie aplikacji na poziomie kodu i zapytań do bazy danych
    ii. testy integracyjne - aby sprawdzić, czy poszczególne komponenty współgrają ze sobą
    iii. testy funkcjonalne - aby sprawdzić działanie poszczególnych funkcji aplikacji
    iv. testy wydajności - aby zapewnić szybkość łączenia z bazą danych i poprawne działanie aplikacji nawet przy dużym obciążeniu
    v. testy kompatybilności - aby zapewnić działanie aplikacji na różnych urządzeniach i systemach
    vi. testy bezpieczeństwa - aby zapewnić bezpieczeństwo danych użytkowników
e. Zgłaszanie błędów przez użytkowników
    i. wybranie kanału zgłaszania błędów
    ii. nadanie priorytetu problemom
    iii. regularna analiza zgłaszanych błędów i przygotowywanie raportów
f. Utrzymanie jakości wytwarzanego oprogramowania
    i. zapewnienie rozwoju członków zespołu poprzez szkolenia i doskonalenie umiejętności
    ii. określanie kryteriów sukcesu oraz standardów jakości dla poszczególnych wymagań
    iii. spotkania zespołu w celu weryfikacji postępów, spotkania z interesariuszami w celu określenia wyzwań i problemów do rozwiązania

5. PLAN WYKONANIA

Plan bazy danych



Rozplanowanie ekranów aplikacji

Link do tablicy z rozplanowaniem:
https://miro.com/app/board/uXjVN7uRoRU=/?share_link_id=586550889593

6. OCENA PRACOCHŁONNOŚCI

Metoda punktów funkcyjnych

Kategoria	Niska złożoność	Średnia złożoność	Wysoka złożoność
Zewnętrzne wejścia (EI)	7	3	1
Zewnętrzne wyjścia (EO)	6	1	1
Logiczne wewnętrzne typy plików (ILF)	2	6	0
Zewnętrzne typy interfejsów (EIF)	1	1	0
Zewnętrzne typy zapytań (EE)	4	1	0

UFP = 778
TDI = 32
AFP = 754,66

Java: 53 LOC/AFP
LOC = 39 996,98

COCOMO

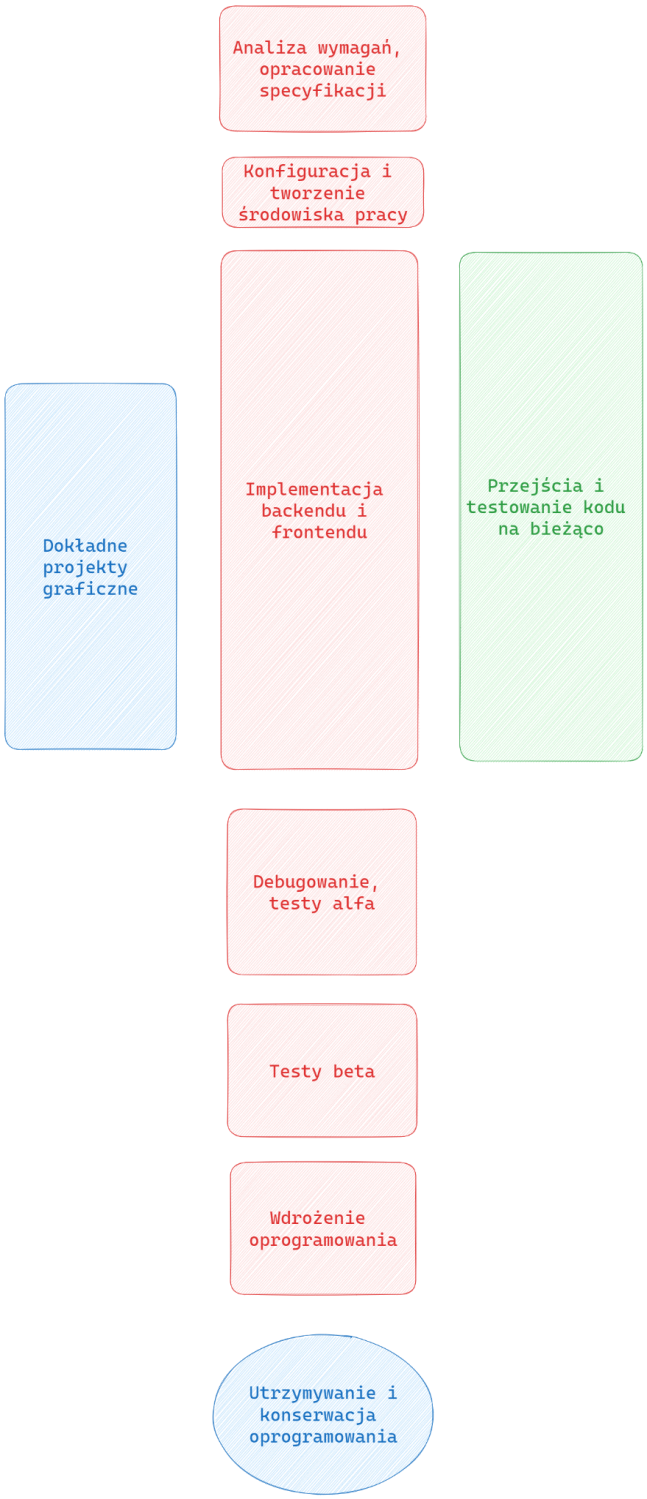
1 KDSI = 1000 LOC
KDSI = 39,99698
Złożoność projektu: łatwy ("organic mode")

Pracochłonność:

Nakład pracy w osobomiesiącach: 115.436
Czas potrzebny do rozwoju projektu [miesiące]: 11.4259
Liczba osób, przy której projekt będzie najefektywniej zrealizowany: 10

7. HARMONOGRAM

a. Analiza wymagań, opracowanie specyfikacji: 1 miesiąc
b. Konfiguracja i tworzenie środowiska pracy: 0.5 miesiąca
c. Implementacja backendu i frontendu: 4 miesiące
d. Dokładne projekty graficzne: 3 miesiące
e. Przejścia i testowanie kodu na bieżąco: 4 miesiące
f. Debugowanie, testy alfa: 1.5 miesiąca
g. Testy beta: 1 miesiąc
h. Wdrożenie oprogramowania: 1 miesiąc
i. Utrzymywanie i konserwacja oprogramowania: na bieżąco



8. OCENA ZGODNOŚCI Z WIZJĄ SYSTEMU I SPECYFIKACJĄ WYMAGAŃ

Po stworzeniu odpowiedniego planu i dokumentacji wprowadziłyśmy zmianę w zakresie wizji oprogramowania. Zrezygnowałyśmy z udostępniania wewnętrznej formy kontaktu pomiędzy użytkownikami - jest to funkcja, którą bez problemu można przenieść do istniejących już komunikatorów, aby skupić się na dostarczeniu aplikacji jak najlepszej jakości. W związku z tym uaktualniłyśmy szacunkowy czas realizacji projektu, także biorąc pod uwagę wymagający proces, jakim jest testowanie aplikacji. Zakładamy, że te plany będą uaktualniane na bieżąco w trakcie prac, ponieważ niektóre etapy mogą zostać zakończone wcześniej, a inne wymagać więcej uwagi i dokładności.