Dobór i adaptacja metodyki

Analiza i Rekomendacje dla Projektu Systemu STOS

Wersja 1.0.0
Data utworzenia obecnej wersji: 07-06-2025
Data utworzenia dokumentu: 07-06-2025

Mateusz Fydrych
193410

Damian Trowski
193443

Maciej Raciniewski
189774

Spis treści

1 O projekcie i produkcie

Projekt: Opracowanie podsystemów sprawdzających zadania dla różnych języków programowania dla Systemu STOS.

Produkt: System automatycznego sprawdzania zadań programistycznych w różnych językach programowania (C, Python, Rust) zrealizowany w formie oddzielnych kontenerów Docker.

Nasz projekt dotyczy stworzenia modularnego systemu do automatycznego sprawdzania zadań programistycznych dla studentów. System STOS będzie składał się z niezależnych kontenerów Docker odpowiedzialnych za kompilację, uruchamianie i ocenianie programów napisanych w różnych językach programowania. Głównym celem jest usprawnienie procesu oceniania prac studentów poprzez automatyzację sprawdzania poprawności rozwiązań oraz zapewnienie jednolitych kryteriów oceny.

Proces sprawdzania zadania będzie przebiegał następująco:

- Kompilacja kodu źródłowego w odpowiednim kontenerze dla danego języka
- Uruchomienie skompilowanego programu z różnymi danymi wejściowymi
- Zapisanie wyników działania programu
- Porównanie wyników z oczekiwanymi odpowiedziami
- Ocena rozwiązania na podstawie poprawności wyników i dodatkowych kryteriów

Modularność rozwiązania pozwoli na łatwe rozszerzanie systemu o nowe języki programowania oraz efektywne skalowanie całego rozwiązania.

2 Ocena według modelu uproszczonego - 5 kryteriów

2.1 Rozmiar

Ocena: Mały do średniego

Uzasadnienie: Nasz projekt realizowany jest przez zespół 3-osobowy, co klasyfikuje go jako mały pod względem liczby osób. Zakres funkcjonalny obejmuje stworzenie kontenerów dla trzech języków programowania (C, Python, Rust) oraz kontenera oceniającego, co stanowi umiarkowaną złożoność. Całkowita liczba story points w backlogu produktu wynosi 115, co wskazuje na średni rozmiar projektu.

Wskazanie: Preferencja dla metodyk zwinnych, które dobrze sprawdzają się w małych zespołach pracujących nad projektami o umiarkowanej złożoności.

2.2 Krytyczność

Ocena: Średnia (kluczowe fundusze)

Uzasadnienie: System będzie używany do automatycznej oceny zadań programistycznych studentów, co wiąże się z odpowiedzialnością za sprawiedliwe i dokładne ocenianie. Awaria systemu lub błędy w ocenianiu mogą prowadzić do nieprawidłowych ocen i frustracji wśród studentów. Jednakże, nie jest to system krytyczny dla życia lub bezpieczeństwa. Ewentualne błędy mogą być skorygowane przez wykładowców poprzez ręczną weryfikację.

Wskazanie: Umiarkowana preferencja dla metodyk zwinnych z elementami dyscypliny w zakresie testowania i zapewnienia jakości.

2.3 Dynamika

Ocena: Średnia

Uzasadnienie: Podstawowe wymagania dotyczące funkcjonalności systemu są dość stabilne - kompilacja kodu, uruchamianie z różnymi danymi wejściowymi i ocenianie wyników. Jednak szczegóły implementacji, takie jak obsługa różnych wersji języków programowania czy mechanizmy bezpieczeństwa, mogą ewoluować w trakcie projektu. Dodatkowo, po wdrożeniu pierwszego języka (C), pojawiły się nowe spostrzeżenia wpływające na implementację pozostałych języków.

Wskazanie: Preferencja dla metodyk zwinnych, które umożliwiają adaptację do zmieniających się wymagań szczegółowych.

2.4 Osoby

Ocena: Korzystna dla zwinności

Uzasadnienie: Nasz zespół składa się z trzech deweloperów z doświadczeniem w programowaniu i konteneryzacji. Wszyscy członkowie zespołu są na poziomie 2-3 według klasyfikacji Boehma i Turnera

(potrafią dopasować metodykę do znanych sytuacji lub przedefiniować ją w nowych sytuacjach). Zespół jest samoorganizujący się i ma doświadczenie w pracy z metodykami zwinnymi.

Wskazanie: Zdecydowana preferencja dla metodyk zwinnych, które wykorzystują umiejętności i samodzielność zespołu.

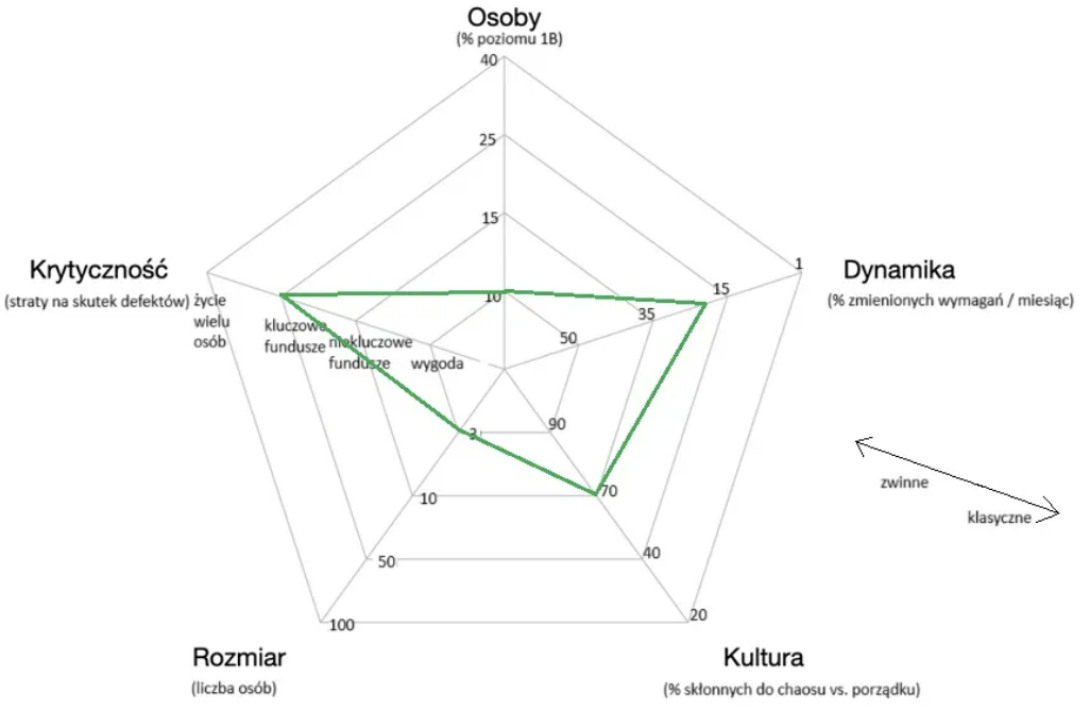
2.5 Kultura

Ocena: Skłonność do chaosu (70%)

Uzasadnienie: Zespół preferuje elastyczne podejście do rozwoju oprogramowania, ceni sobie autonomię i możliwość szybkiego reagowania na zmiany. Członkowie zespołu są otwarci na eksperymenty i innowacje, a nie ścisłe przestrzeganie procedur. Projekt realizowany jest w środowisku akademickim, które sprzyja kreatywności i eksperymentowaniu. Najważniejszym celem jest zadowalający rezultat, a nie sztywno wcześniej założone cele.

Wskazanie: Zdecydowana preferencja dla metodyk zwinnych, które zapewniają swobodę działania i elastyczność.

2.6 Podsumowanie modelu uproszczonego



Rysunek 1: Model uproszczony - ocena projektu STOS

Na podstawie analizy pięciu kryteriów modelu uproszczonego, nasz projekt wyraźnie pasuje do metodyk zwinnych. Wszystkie kryteria wskazują na preferencję dla podejścia zwinnego, szczególnie ze względu na mały rozmiar zespołu, blisko średniej krytyczność systemu, umiarkowaną dynamikę wymagań, wysokie kompetencje zespołu oraz kulturę sprzyjającą elastyczności.

3 Ocena według zaadaptowanego modelu pełnego - 7 kryteriów

3.1 Zastosowanie

Główne cele

Ocena projektu: Głównym celem projektu jest stworzenie efektywnego systemu automatycznego sprawdzania zadań programistycznych, który usprawni proces oceniania i zapewni jednolite kryteria. Projekt ma dostarczyć wartość zarówno dla wykładowców (oszczędność czasu), jak i dla studentów (szybka informacja zwrotna).

Uzasadnienie: Projekt wymaga iteracyjnego podejścia, ponieważ pełna funkcjonalność systemu będzie rozwijana stopniowo, począwszy od obsługi języka C, przez Python, aż po Rust. Możliwość wczesnego testowania i dostosowywania systemu na podstawie doświadczeń z pierwszymi implementacjami jest kluczowa.

Dopasowanie: Metodyki zwinne, umożliwiające iteracyjne dostarczanie wartości i zbieranie informacji zwrotnej.

Środowisko

Ocena projektu: Projekt realizowany jest w środowisku akademickim, gdzie istnieje potrzeba elastycznego dostosowania do różnych wymagań dydaktycznych. System będzie używany przez wykładowców i studentów, którzy mają różne poziomy umiejętności technicznych.

Uzasadnienie: Środowisko akademickie sprzyja eksperymentowaniu i innowacjom, ale wymaga również niezawodności i sprawiedliwości w ocenianiu. Konieczne jest zrozumienie potrzeb obu grup użytkowników i dostosowanie systemu do ich oczekiwań.

Dopasowanie: Metodyki zwinne, które kładą nacisk na współpracę z użytkownikami i adaptację do ich potrzeb.

3.2 Zarządzanie

Komunikacja

Ocena projektu: Zespół jest mały (3 osoby), co sprzyja bezpośredniej i częstej komunikacji. Członkowie zespołu używają narzędzi takich jak Discord do codziennej komunikacji oraz GitHub do śledzenia postępów i zarządzania kodem. Komunikacja z interesariuszami odbywa się regularnie.

Uzasadnienie: Mały zespół może efektywnie komunikować się bez formalnych struktur, co jest charakterystyczne dla metodyk zwinnych. Jednocześnie, dokumentacja techniczna i specyfikacje są ważne dla zapewnienia kompatybilności z systemem STOS.

Dopasowanie: Metodyki zwinne z elementami dokumentacji technicznej.

3.3 Techniczne

Wymagania

Ocena projektu: Ogólne wymagania funkcjonalne są jasno określone (kompilacja, uruchamianie, ocenianie), ale szczegóły implementacyjne mogą się zmieniać w trakcie projektu. Backlog produktu zawiera elementy o różnych priorytetach, co umożliwia elastyczne planowanie prac.

Uzasadnienie: Podejście iteracyjne pozwoli na doprecyzowanie wymagań szczegółowych w trakcie implementacji pierwszych komponentów systemu. Doświadczenia zdobyte przy implementacji obsługi języka C będą cenne przy pracy nad pozostałymi językami.

Dopasowanie: Metodyki zwinne, które umożliwiają ewolucję wymagań i priorytetyzację prac.

Wytwarzanie

Ocena projektu: Projekt wykorzystuje nowoczesne technologie konteneryzacji (Docker), co umożliwia modularną budowę systemu. Każdy kontener jest niezależnym komponentem, który może być rozwijany i testowany oddzielnie.

Uzasadnienie: Modularność rozwiązania sprzyja iteracyjnemu i przyrostowemu podejściu do wytwarzania. Możliwość niezależnego rozwijania poszczególnych kontenerów ułatwia równoległą pracę zespołu.

Dopasowanie: Metodyki zwinne, które wspierają modularną architekturę i przyrostowe dostarczanie funkcjonalności.

3.4 Osoby

Klient

Ocena projektu: Klientami są wykładowcy, którzy będą używać systemu do oceniania zadań studentów. Mają oni jasno określone oczekiwania dotyczące funkcjonalności systemu, ale mogą nie być dostępni na co dzień do konsultacji.

Uzasadnienie: Zaangażowanie klienta jest ważne dla zapewnienia, że system spełnia ich potrzeby, ale nie wymaga codziennej obecności. Regularne demonstracje postępów i zbieranie informacji zwrotnej będą wystarczające.

Dopasowanie: Metodyki zwinne z adaptacją do ograniczonej dostępności klienta.

Kultura

Ocena projektu: Zespół preferuje elastyczne podejście do rozwoju oprogramowania, ceni sobie autonomię i możliwość szybkiego reagowania na zmiany. Środowisko akademickie sprzyja innowacjom i eksperymentowaniu.

Uzasadnienie: Kultura zespołu i organizacji sprzyja adaptacji i innowacji, co jest charakterystyczne dla metodyk zwinnych. Jednocześnie, akademicki charakter projektu wymaga pewnego poziomu formalizacji i dokumentacji.

Dopasowanie: Metodyki zwinne z elementami formalizacji wymaganymi w środowisku akademickim.

3.5 Podsumowanie oceny pełnej

Analiza siedmiu kryteriów w modelu pełnym potwierdza, że nasz projekt najlepiej pasuje do metodyk zwinnych. Wszystkie kryteria wskazują na preferencję dla podejścia zwinnego, z niewielkimi adaptacjami uwzględniającymi specyfikę środowiska akademickiego i ograniczoną dostępność klienta. Szczególnie istotne jest iteracyjne i przyrostowe podejście do wytwarzania, które umożliwi stopniowe rozwijanie funkcjonalności systemu i dostosowywanie go do potrzeb użytkowników.

4 Model dostarczania produktu końcowego projektu

Na podstawie analizy charakterystyki projektu oraz wniosków z poprzednich ocen, rekomendujemy **przyrostowy model dostarczania** (Incremental Delivery) jako najlepiej dopasowany do naszego projektu.

Uzasadnienie wyboru modelu przyrostowego

- **Modularność systemu** - Projekt składa się z niezależnych kontenerów dla różnych języków programowania, które mogą być rozwijane i dostarczane oddzielnie.
- **Priorytetyzacja funkcjonalności** - Backlog produktu zawiera elementy o różnych priorytetach, co umożliwia dostarczanie najpierw najważniejszych funkcjonalności.
- **Wartość biznesowa na wczesnym etapie** - Model przyrostowy pozwoli na wczesne dostarczenie wartości biznesowej poprzez implementację obsługi pierwszego języka programowania (C), co umożliwi rozpoczęcie testowania systemu przez użytkowników.
- **Redukcja ryzyka** - Wczesne dostarczenie części funkcjonalności pozwoli na identyfikację potencjalnych problemów i dostosowanie dalszych prac.

Przykładowy plan przyrostów

- **Przyrost 1: Podstawowa infrastruktura i obsługa języka C**
  - Kontener oceniający (PB01)
  - Skrypty wyświetlania wyników (PB02)
  - Obsługa języka C (PB03)
  - Testy obrazów (PB06)
- **Przyrost 2: Rozszerzenie o język Python i integracja z systemem STOS**
  - Obsługa języka Python (PB04)
  - Implementacja obrazu kompatybilnego z STOS (PB07)
- **Przyrost 3: Obsługa języka Rust i automatyzacja procesów**
  - Obsługa języka Rust (PB05)
  - Testowanie w CI/CD (PB08)
  - CI/CD pipeline dla DockerHub (PB09)

Sugerowana metodyka wspierająca model przyrostowy

Na podstawie przeprowadzonych analiz, rekomendujemy **metodykę Kanban** jako najlepiej dopasowaną do naszego projektu. Kanban wspiera model przyrostowy dostarczania produktu, umożliwiając elastyczne zarządzanie pracą i ciągłe dostarczanie wartości.

Uzasadnienie wyboru Kanban

- **Elastyczność** - Kanban nie narzuca sztywnych ram czasowych (sprintów), co pozwala na elastyczne dostosowywanie się do zmieniających się priorytetów.

- **Wizualizacja przepływu pracy** - Tablica Kanban zapewnia przejrzystość procesu i umożliwia łatwe śledzenie postępów.
- **Ograniczenie pracy w toku (WIP)** - Kanban pozwala na kontrolowanie ilości równoczesnych zadań, co zwiększa efektywność zespołu.
- **Ciągłe doskonalenie** - Metodyka zachęca do regularnej analizy procesu i wprowadzania usprawnień.
- **Minimalizacja formalności** - Kanban wymaga mniej ceremonii niż Scrum, co jest odpowiednie dla małego zespołu.

5 Metodyka i jej adaptacja

5.1 Podsumowanie sugestii metodyki

Analiza projektu zarówno według modelu uproszczonego (5 kryteriów), jak i pełnego (7 kryteriów), a także preferowany model dostarczania produktu (przyrostowy), jednoznacznie wskazują na metodyki zwinne jako najbardziej odpowiednie dla naszego projektu. Spośród dostępnych metodyk zwinnych, **Kanban** został wybrany jako bazowa metodyka ze względu na jego elastyczność, minimalizm formalny i doskonałe dopasowanie do modelu przyrostowego dostarczania produktu.

5.2 Wybór konkretnej metodyki: Kanban

Kanban jest lekką metodyką zwinną, która koncentruje się na wizualizacji przepływu pracy, ograniczeniu pracy w toku (WIP) oraz ciągłym doskonaleniu procesu. W przeciwieństwie do Scruma, Kanban nie narzuca sztywnych ram czasowych (sprintów), co zapewnia większą elastyczność w dostosowywaniu się do zmieniających się priorytetów.

Kluczowe zasady Kanban, które będziemy stosować

- **Wizualizacja przepływu pracy** - Tablica Kanban z kolumnami reprezentującymi etapy procesu (Do zrobienia, W trakcie, Przegląd kodu, Testowanie, Gotowe)
- **Ograniczenie pracy w toku (WIP)** - Limit zadań w kolumnie "W trakcie"do maksymalnie 2 na osobę
- **Zarządzanie przepływem** - Monitorowanie i optymalizacja przepływu pracy
- **Jawne zasady procesu** - Jasno określone kryteria przejścia między etapami
- **Ciągłe doskonalenie** - Regularne spotkania retrospektywne

5.3 Adaptacja metodyki Kanban

Mimo ogólnego dopasowania do Kanban, nasz projekt posiada cechy, które wymagają świadomej adaptacji metodyki:

5.3.1 Akademicki charakter projektu

Niedopasowanie: Kanban nie zakłada formalnego raportowania postępów, które może być wymagane w środowisku akademickim.

Adaptacja:

- Wprowadzenie regularnych odbywających się co dwa tygodnie spotkań z opiekunem projektu
- Utrzymywanie aktualnej dokumentacji projektu w repozytorium GitHub

5.3.2 Potrzeba zapewnienia wysokiej jakości i bezpieczeństwa

Niedopasowanie: Standardowy Kanban nie kładzie szczególnego nacisku na formalne procesy zapewnienia jakości, które są istotne w systemie oceniającym prace studentów.

Adaptacja:

- Dodanie dedykowanych kolumn na tablicy Kanban dla "Do recenzji"i "Testowania"
- Automatyzacja testów jednostkowych, integracyjnych i bezpieczeństwa w pipeline CI/CD

5.3.3 Potrzeba dokumentacji technicznej

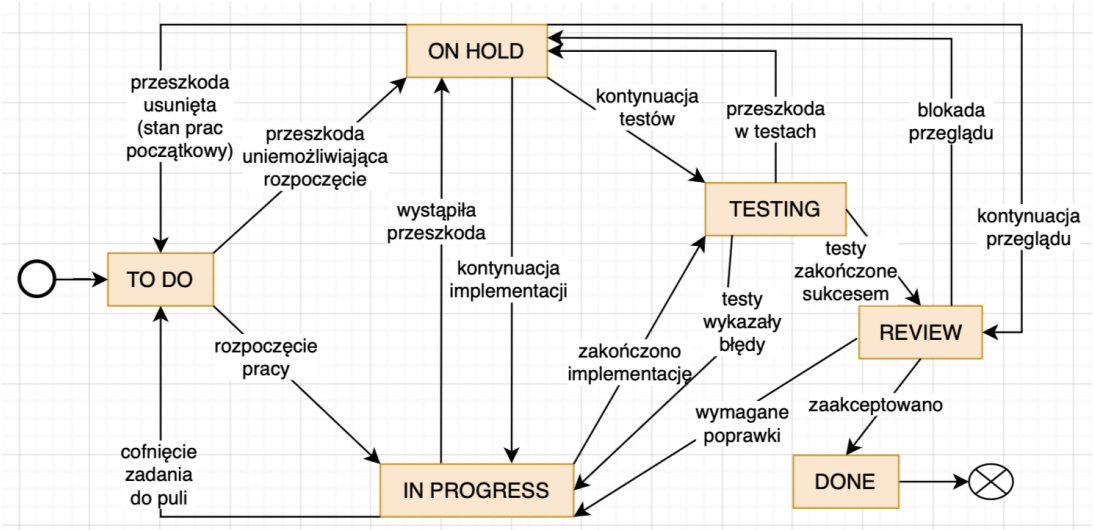
Niedopasowanie: Kanban, jako metodyka zwinna, preferuje "działające oprogramowanie ponad obszerną dokumentację", jednak w naszym projekcie dokumentacja techniczna jest istotna dla przyszłych użytkowników i rozwijających system.

Adaptacja:

- Włączenie zadań dokumentacyjnych do tablicy Kanban jako równorzędnych z zadaniami programistycznymi
- Wykorzystanie narzędzi automatycznego generowania dokumentacji z kodu (prawdopodobnie Doxygen)
- Przygotowanie kompleksowej dokumentacji użytkownika i administratora systemu

5.4 Praktyczna implementacja adaptowanej metodyki

5.4.1 Tablica Kanban



Rysunek 2: Uproszczony diagram stanów

Tablica Kanban będzie zawierająca następujące kolumny:
- **Do zrobienia** - Zadania wybrane do realizacji w najbliższym czasie
- **W zatrzymaniu** - Zadanie zostało wstrzymane
- **W trakcie** - Zadania aktualnie realizowane
- **Testowanie** - Zadania w trakcie testowania
- **Do recenzji** - Zadania, które wymagają oceny innego członka zespołu
- **Gotowe** - Zadania ukończone

5.4.2 Spotkania

Mimo że Kanban nie narzuca regularnych spotkań, w naszej adaptacji wprowadzimy:
- **Przegląd tablicy (2 razy w tygodniu, 30 min)** - Analiza przepływu zadań, identyfikacja wąskich gardeł i synchronizacja pracy
- **Retrospektywa (co 2 tygodnie, 1h)** - Analiza procesu i identyfikacja obszarów do usprawnienia

5.4.3 Narzędzia

Do implementacji naszej adaptowanej metodyki Kanban wykorzystamy:
- **GitHub Projects** - Do zarządzania tablicą Kanban i backlogiem produktu
- **GitHub Actions** - Do automatyzacji testów i wdrożeń (CI/CD)
- **Discord** - Do codziennej komunikacji zespołu
- **DockerHub** - Do publikacji obrazów Docker

5.5 Podsumowanie adaptacji

Zaproponowana adaptacja metodyki Kanban zachowuje jej kluczowe zalety (elastyczność, wizualizacja przepływu pracy, ograniczenie WIP), jednocześnie adresując specyficzne wymagania naszego projektu związane z akademickim charakterem, integracją z istniejącym systemem, zapewnieniem jakości i bezpieczeństwa oraz potrzebą dokumentacji technicznej.

Dzięki tej adaptacji, będziemy w stanie efektywnie zarządzać projektem, dostarczać wartość w modelu przyrostowym oraz reagować na zmieniające się priorytety i wymagania. Jednocześnie, zachowamy

niezbędny poziom formalizacji i dokumentacji wymagany w środowisku akademickim i przy tworzeniu systemu do automatycznego oceniania zadań programistycznych.