Uniwersytet Gdański Wydział Matematyki, Fizyki i Informatyki Instytut Informatyki

Oliver Gruba, Maciej Nasiadka

16 stycznia 2026

Imię i Nazwisko (nr indeksu)	Oliver Gruba (292583) Maciej Nasiadka (292574)
Nazwa uczelni	Uniwersytet Gdański
Kierunek	Informatyka (profil praktyczny)
Prowadzący	dr inż. Stanisław Witkowski
Nazwa ćwiczenia	Wymagania funkcjonalne i niefunkcjonalne w projektowaniu systemów rozproszonych.
Numer sprawozdania	11
Data zajęć	15.01.2026
Data oddania	21.01.2026
Miejsce na ocenę	

Spis treści

1. Wstęp

Fundamentem każdej poprawnie zaprojektowanej architektury oprogramowania jest precyzyjna inżynieria wymagań. W erze systemów rozproszonych i mikrousług, rozróżnienie między tym, *co* system robi, a tym, *jak* to robi, determinuje sukces projektu. Niniejsze sprawozdanie systematyzuje wiedzę na temat wymagań funkcjonalnych i niefunkcjonalnych, ze szczególnym uwzględnieniem ich wpływu na decyzje architektoniczne.

W praktyce inżynierskiej to właśnie błędnie zdefiniowane wymagania są najczęstszą przyczyną porażek projektów informatycznych. Statystyki branżowe wskazują, że znaczący odsetek projektów przekracza budżet lub harmonogram właśnie z powodu niejasnych, sprzecznych albo niekompletnych wymagań. Brak precyzji na etapie analizy skutkuje koniecznością kosztownych zmian w późnych fazach cyklu życia systemu.

Wymagania pełnią rolę kontraktu pomiędzy zamawiającym a zespołem wytwórczym. Są punktem odniesienia dla analityków, projektantów, programistów, testerów oraz interesariuszy biznesowych. Im wcześniej zostaną poprawnie sformułowane i zweryfikowane, tym mniejsze ryzyko, że końcowy produkt nie spełni oczekiwań użytkowników.

2. Definicje i klasyfikacja wymagań

2.1. Co to są wymagania funkcjonalne?

Wymagania funkcjonalne (ang. *Functional Requirements*) stanowią opis zachowania systemu. Definiują funkcje, procesy i usługi, które system musi dostarczyć, aby zaspokoić potrzeby biznesowe użytkownika. Są one zazwyczaj wyrażane w formie zdań twierdzących ("System powinien...") lub przypadków użycia (Use Cases).

W praktyce wymagania funkcjonalne opisują cały „scenariusz życia" systemu: od momentu wejścia użytkownika do aplikacji, poprzez wykonywanie przez niego konkretnych operacji, aż po zapis i przetwarzanie danych. Każde wymaganie funkcjonalne powinno odpowiadać na realną potrzebę biznesową – funkcja, która nie wnosi wartości, generuje jedynie koszt utrzymania.

Źle sformułowane wymagania funkcjonalne prowadzą do tzw. funkcjonalności pozornych – system coś robi, ale nie to, czego faktycznie potrzebuje użytkownik. Dlatego ważne jest, aby wymagania były:

- jednoznaczne,
- mierzalne,
- testowalne,

- zrozumiałe dla nietechnicznych interesariuszy.

2.2. Co to są wymagania niefunkcjonalne?

Wymagania niefunkcjonalne (ang. *Non-functional Requirements*, NFR), często nazywane atrybutami jakościowymi (ang. *Quality Attributes*), określają ograniczenia i standardy działania systemu. Nie wprowadzają nowych funkcji, lecz narzucają ramy, w jakich funkcje te muszą być realizowane.

W praktyce to właśnie wymagania niefunkcjonalne decydują o tym, czy system będzie używalny w realnych warunkach. Użytkownik może zaakceptować brak pewnych funkcji, ale rzadko zaakceptuje system, który:

- działa wolno,
- często się zawiesza,
- traci dane,
- nie chroni prywatności.

Wymagania niefunkcjonalne są często trudniejsze do uchwycenia, ponieważ interesariusze rzadko formułują je wprost. Zamiast tego mówią: „system ma działać płynnie", „ma być bezpieczny", „ma się nie psuć". Zadaniem analityka jest przełożenie tych ogólnych oczekiwań na mierzalne parametry techniczne.

3. Analiza porównawcza: funkcjonalne vs niefunkcjonalne

Rozróżnienie między tymi dwoma typami wymagań nie zawsze jest oczywiste, jednak kluczowe dla procesu deweloperskiego. Poniższa tabela przedstawia wielowymiarowe porównanie.

Tabela 1: Szczegółowa analiza różnic

Kryterium	Wymagania funkcjonalne	Wymagania niefunkcjonalne
Pytanie kluczowe	**Co** system ma robić?	**Jak** system ma działać?
Natura	Binarne (Działa / Nie działa). Funkcja jest albo wdrożona, albo nie.	Ciągła (Skala). System może być "mało wydajny"lub "bardzo wydajny".
Metoda weryfikacji	Testy jednostkowe, integracyjne, scenariusze biznesowe (User Stories).	Benchmarki, testy obciążeniowe, audyty bezpieczeństwa, testy penetracyjne.
Odbiorca	Użytkownik końcowy (End User).	Architekt systemu, DevOps, Administrator.
Koszt zmiany	Relatywnie niski (zmiana w kodzie jednej klasy/funkcji).	**Bardzo wysoki** (często wymaga zmiany architektury, np. wymiany bazy danych).
Źródło	Potrzeby biznesowe klienta.	Ograniczenia techniczne, prawne (RODO), standardy branżowe.

4. Zastosowania i wpływ na architekturę IT

Zrozumienie wymagań niefunkcjonalnych jest kluczowym zadaniem architekta systemów, ponieważ to one, a nie funkcje, dyktują kształt architektury (tzw. *Architectural Drivers*). Funkcje można zazwyczaj zaimplementować na wiele sposobów, jednak sposób ich realizacji jest silnie ograniczony przez wymagania jakościowe.

Przykładowo, system bankowy i system blogowy mogą posiadać podobne funkcje (logowanie, zarządzanie kontem), lecz ich architektury będą zupełnie inne ze względu na odmienne wymagania dotyczące bezpieczeństwa, niezawodności i dostępności.

Architektura systemu jest więc kompromisem pomiędzy:

- potrzebami biznesowymi,
- ograniczeniami technologicznymi,
- kosztami utrzymania,
- ryzykiem operacyjnym.

Błędem jest projektowanie architektury wyłącznie na podstawie wymagań funkcjonalnych – prowadzi to do rozwiązań, które działają poprawnie w środowisku testowym, ale zawodzą w rzeczywistej eksploatacji.

5. Przykłady wymagań

5.1. Przykłady wymagań funkcjonalnych

- **WF-01 Koszyk:** System musi automatycznie przeliczać wartość koszyka po dodaniu produktu, uwzględniając rabaty.
- **WF-02 Raportowanie:** Administrator ma możliwość eksportu listy użytkowników do pliku CSV.
- **WF-03 API:** System udostępnia endpoint REST API '/api/orders' zwracający historię zamówień w formacie JSON.

5.2. Przykłady wymagań niefunkcjonalnych

- **WNF-01 Wydajność (Latency):** Czas generowania strony głównej nie może przekroczyć 200ms dla 95% zapytań (95th percentile).
- **WNF-02 Skalowalność:** System musi obsługiwać 10 000 jednoczesnych użytkowników bez błędów typu 5xx.
- **WNF-03 Dostępność (Availability):** Przerwa w działaniu serwisu nie może być dłuższa niż 4 godziny rocznie (SLA 99.95%).
- **WNF-04 Bezpieczeństwo:** Wszystkie hasła muszą być haszowane algorytmem Argon2 lub bcrypt z solą.

6. Zalety precyzyjnego definiowania wymagań

6.1. Zalety wymagań funkcjonalnych

- Pozwalają na jasne zdefiniowanie **zakresu prac** (Scope of Work), chroniąc projekt przed niekontrolowanym rozrostem (scope creep).
- Są podstawą do tworzenia scenariuszy testowych UAT, które decydują o odbiorze projektu przez klienta.

6.2. Zalety wymagań niefunkcjonalnych

- Zapewniają **doświadczenie użytkownika (UX)**. Nawet najbardziej funkcjonalny system zostanie odrzucony, jeśli będzie działał wolno.
- Redukują **dług techniczny**. Wczesne uwzględnienie wymagań np. modyfikowalności kodu zapobiega konieczności przepisywania systemu po roku.

- Gwarantują zgodność z prawem (np. wymogi dotyczące retencji danych).

7. Pozyskiwanie wymagań (elicitation)

Proces pozyskiwania wymagań polega na identyfikacji rzeczywistych potrzeb interesariuszy systemu. Jest to proces iteracyjny – wymagania są stopniowo doprecyzowywane wraz ze wzrostem wiedzy o problemie i ograniczeniach technicznych.

Jednym z największych zagrożeń jest tzw. „efekt pierwszej wersji" – interesariusze często opisują rozwiązanie, a nie problem. Rolą analityka jest dotarcie do rzeczywistej potrzeby biznesowej, a nie jedynie spisanie pierwszej zaproponowanej koncepcji.

Proces elicitation powinien uwzględniać różne perspektywy:

- użytkownika końcowego,
- właściciela produktu,
- zespołu technicznego,
- działu prawnego i bezpieczeństwa.

8. Dokumentowanie wymagań

Dobrze opisane wymagania powinny być jednoznaczne, testowalne i zrozumiałe.

8.1. Format opisu

- identyfikator wymagania,
- opis w języku naturalnym,
- kryteria akceptacji,
- priorytet biznesowy.

8.2. Narzędzia

- dokumenty SRS,
- backlog produktowy,
- narzędzia typu Jira, Confluence, Trello.

9. Priorytetyzacja wymagań

Nie wszystkie wymagania mają taką samą wagę biznesową.

9.1. Metody priorytetyzacji

- MoSCoW (Must, Should, Could, Won't),
- analiza wartości biznesowej,
- analiza ryzyka.

10. Walidacja i weryfikacja wymagań

10.1. Walidacja

Walidacja odpowiada na pytanie: „Czy budujemy właściwy system?". Polega na sprawdzeniu, czy wymagania rzeczywiście odzwierciedlają potrzeby użytkownika i cel biznesowy projektu. Najczęściej realizowana jest poprzez:

- przeglądy z interesariuszami,
- makiety i prototypy,
- scenariusze użycia.

10.2. Weryfikacja

Weryfikacja odpowiada na pytanie: „Czy wymagania są poprawnie zapisane?". Sprawdza się:

- spójność pomiędzy wymaganiami,
- brak sprzeczności,
- możliwość testowania,
- wykonalność techniczną.

Brak walidacji prowadzi do tworzenia systemów, które są poprawne technicznie, ale bezużyteczne biznesowo. Brak weryfikacji skutkuje chaosem implementacyjnym i błędami projektowymi.

11. Typowe błędy w definiowaniu wymagań

- zbyt ogólne opisy ("system ma być szybki"),
- brak kryteriów akceptacji,
- mieszanie wymagań funkcjonalnych z niefunkcjonalnymi,
- ignorowanie interesariuszy technicznych.

12. Opis wymagań dla systemu "FoodDelivery"(case study)

12.1. Kontekst

Projekt dotyczy systemu zamawiania jedzenia online. System składa się z aplikacji mobilnej, panelu restauratora oraz backendu w architekturze mikrousług.

12.1.1 Wymagania funkcjonalne (co robi system?)

- [**WF-ORDER**] Użytkownik może filtrować restauracje po typie kuchni (włoska, azjatycka) oraz ocenie (gwiazdki).
- [**WF-PAY**] System integruje się z bramką płatności BLIK i przetwarza callback o statusie transakcji.
- [**WF-NOTIFY**] System wysyła powiadomienie Push do klienta, gdy kurier znajdzie się w promieniu 500m od celu.

12.1.2 Wymagania niefunkcjonalne (jakość i ograniczenia)

- [**WNF-PERF**] System musi obsłużyć nagły skok ruchu w porze lunchowej (12:00-14:00) zwiększając zasoby obliczeniowe w chmurze w czasie poniżej 2 minut (Autoscaling).
- [**WNF-SEC**] Komunikacja między mikrousługami musi być szyfrowana (mTLS).
- [**WNF-RELIABILITY**] W przypadku awarii serwisu płatności, użytkownik nie traci zawartości koszyka, a system ponawia próbę płatności (Circuit Breaker pattern).

13. Wnioski

Poprawne zdefiniowanie wymagań funkcjonalnych i niefunkcjonalnych jest kluczem do sukcesu każdego projektu informatycznego.

- **Wymagania funkcjonalne** budują wartość biznesową dla klienta.
- **Wymagania niefunkcjonalne** decydują o stabilności, bezpieczeństwie i skalowalności rozwiązania.

Największym błędem projektowym jest traktowanie wymagań niefunkcjonalnych jako dodatku, który można doprecyzować „później". W praktyce to one powinny być jednym z pierwszych elementów analizy, ponieważ narzucają kierunek całemu projektowi.

Dobrze przygotowany dokument wymagań:

- zmniejsza ryzyko nieporozumień,
- ułatwia estymację kosztów i czasu,
- stanowi podstawę testów i odbioru systemu,
- ogranicza chaotyczne zmiany w trakcie realizacji.

W projektowaniu architektury mikrousługowej wymagania funkcjonalne pomagają identyfikować granice usług, natomiast wymagania niefunkcjonalne determinują sposób komunikacji, strategie skalowania oraz dobór infrastruktury. Ignorowanie ich na wczesnym etapie niemal zawsze prowadzi do kosztownych refaktoryzacji lub całkowitej przebudowy systemu.