Analiza i projektowanie systemów informatycznych

Definiowanie i analiza wymagań

Rodzaje wymagań

- wymagania funkcjonalne
  - określenie funkcjonalności systemu
  - określenie sposobu użycia funkcjonalności przez byty zewnętrzne (użytkownicy, inne systemy)
- wymagania niefunkcjonalne
  - ilościowe
  - jakościowe

Wymagania funkcjonalne

- perspektywa funkcjonalna
  - widok od wewnątrz systemu – określa zestaw funkcji, które system może wykonywać
  - lista funkcji
  - grupy funkcji
  - hierarchia funkcji
- perspektywa użycia
  - widok od zewnątrz systemu – określa sposoby użycia funkcjonalności systemu w realnych przypadkach
  - przypadki użycia
  - aktorzy

Przypadek użycia (Use Case)

- Definicja:
  - Przypadek użycia jest to dokument opisowy objaśniający sekwencje zdarzeń występujących w ramach procesu używania systemu przez „aktora”, czyli obiekt zewnętrzny.
- Własności:
  - Przypadki użycia same nie stanowią specyfikacji wymagań, ale są doskonałym narzędziem do ich ilustracji i weryfikacji.
    - korzystają ze zdefiniowanych funkcji systemu,
    - odnoszą się do innych wymagań (funkcjonalnych i niefunkcjonalnych)
  - Przypadek użycia jest zawsze inicjowany przez aktora
    - nie służy do opisu wewnętrznych działań systemu, ani też działań automatycznych

Przypadek użycia (Use Case)

- Własności:
  - Przypadek użycia jest zawsze opisem „dużego”, kompletnego procesu, zwierającego zwykle wiele kroków lub transakcji, produkującego wynik odbierany przez aktora.
    - Opisuje całość interakcji pomiędzy użytkownikiem, a systemem przy realizacji przez użytkownika pełnego, zamkniętego zadania.
    - Nie jest opisem pojedynczego kroku, wywołania funkcji, itp.

Przypadki użycia – forma opisu

- Opis ogólny:
  - nazwa przypadku użycia,
  - aktorzy,
  - typ,
  - opis,
  - warunki wejściowe i wyjściowe,
  - referencje (lista funkcji, wymagań, etc.)
- Przebieg zdarzeń:
  - typowy,
  - alternatywne,
  - wyjątki
  - opis tekstowy, tabelaryczny,
  - diagramy aktywności

Przebieg zdarzeń

- Opis tabelaryczny

1. inicjacja
2. pierwsza akcja aktora
3. odpowiedź systemu
4. kolejna akcja aktora
5. kolejna odpowiedź systemu
6. kolejna akcja aktora
7. kolejna odpowiedź systemu

   ...

   ...

- Opis liniowy

1. Aktor…
2. System…
3. Aktor…

   …

Przykładowy szablon

Nazwa	
ID	
Aktor główny	
Aktorzy	
Priorytet	
Opis	
Wyzwalanie	
Warunki początkowe	
Przebieg zdarzeń	
Przebiegi alternatywne	
Wyjątki	
Warunki końcowe	
Wymagania funkcjon.	
Wymagania niefunkc.	
Założenia	
Źródła	

Przypadki użycia – podział

- poziom opisu:
  - wysokiego poziomu *(high level)* – tylko opis ogólny
    - tworzone podczas identyfikacji przypadków użycia, na początku zbierania wymagań
  - rozszerzone *(expanded)* – opis ogólny wraz z przebiegiem zdarzeń
    - rozszerzanie opisu podczas dalszego zbierania wymagań szczegółowych i w trakcie analizy
- priorytet, waga:
  - główne *(primary)* – najważniejsze procesy systemu
    - muszą być zrealizowane
    - wśród nich wyróżniamy przypadki architektonicznie znaczące
  - drugorzędne *(secondary)* – procesy mniej istotne lub rzadko wykonywane
    - muszą być zrealizowane, ale w końcowych iteracjach
  - opcjonalne *(optional)* – procesy poboczne, dodatkowe
    - nie muszą być koniecznie realizowane

Przypadki użycia – podział

- wnikliwość opisu:
  - merytoryczne, istotne *(essential)* – koncentrujące się na istocie problemu, opisane w formie wolnej od szczegółów technicznych i implementacyjnych
    - tworzone przez analityków, aby nie ograniczać projektantów w doborze technologii
  - rzeczywiste, konkretne *(real)* – opisane przy wykorzystaniu terminologii technologicznej i zawierające szczegóły implementacyjne (np. mechanizmy realizacji wejścia/wyjścia)
    - tworzone przez projektantów – dobór odpowiedniej technologii i sposobu realizacji przy uwzględnieniu wszystkich wymagań niefunkcjonalnych

Przypadki użycia – podział

- zakres opisu:
  - biznesowe *(business)* – opisujące relacje otoczenia z organizacją
  - *systemowe (system)* – opisujące interakcję obiektów zewnętrznych z systemem
    - interfejsowe *(black box)* – tradycyjne, opisujące interakcję systemu z otoczeniem, BEZ opisu jakichkolwiek działań wewnętrznych systemu
    - wewnętrzne *(white box)* – pozwalające na uwzględnianie wewnętrznych działań systemu
  - przypadki współpracy *(Collaboration Cases)* – użycie techniki przypadków użycia do opisu interakcji pomiędzy elementami systemu

Aktor *(Actor)*

- Definicja:
  - Aktor jest bytem zewnętrznym w stosunku do systemu, wchodzącym w interakcje z systemem: wysyłającym i odbierającym komunikaty, wymieniającym informacje.
- Własności:
  - Aktor reprezentuje rolę grana w danym przypadku użycia:
    - rola człowieka,
    - system komputerowy,
    - urządzenie elektroniczne.
  - Aktor jest klasą, nie instancją
    - Możliwe jest klasyfikowanie aktorów poprzez mechanizm generalizacji-specjalizacji.

Aktorzy – podział

- ranking aktorów:
  - główny *(primary)* – używający zasadniczych funkcji systemu, biorący udział w głównych przypadkach użycia,
  - drugorzędny *(secondary)* – wykorzystujący fukcje pobocze lub administracyjne.
- aktywność:
  - aktywny *(active)* – inicjujący przypadek użycia,
  - pasywny *(passive)* – uczestnik scenariusza odpowiadający na sygnały.

Identyfikacja przypadków użycia

- poprzez aktorów:
  - identyfikacja aktorów,
  - dla każdego aktora – identyfikacja procesów i funkcji, które inicjują lub w których biorą udział (odczyt, tworzenie, aktualizacja, usuwanie danych, informowane o zdarzeniach).
- poprzez zdarzenia:
  - identyfikacja zdarzeń zewnętrznych, na które system musi odpowiadać,
  - powiązanie zdarzeń z aktorami i procesami.
- poprzez funkcje:
  - identyfikacja funkcji, które system ma wykonywać,
  - określenie kontekstów (procesów), w których każda funkcja jest wykonywana.

Związki pomiędzy przypadkami użycia

- generalizacja-specjalizacja
  - jeden przypadek użycia stanowi częściowy opis zachowania innego przypadku użycia
  - odpowiednik dziedziczenia w programowaniu
  - pozwala opisywać w przypadku specjalizowanym kroki odziedziczone (niezmienione lub specjalizowane) i nowe
- włączenie <<include>>
  - jeden przypadek użycia (lub wiele) wykorzystuje zawsze i w całości inny przypadek użycia
  - odpowiednik mechanizmu „include” w programowaniu
- rozszerzenie <<extends>>
  - jeden przypadek użycia (lub wiele) wykorzystuje w pewnych sytuacjach inny przypadek użycia
  - odpowiednik wywołania podprogramu w programowaniu

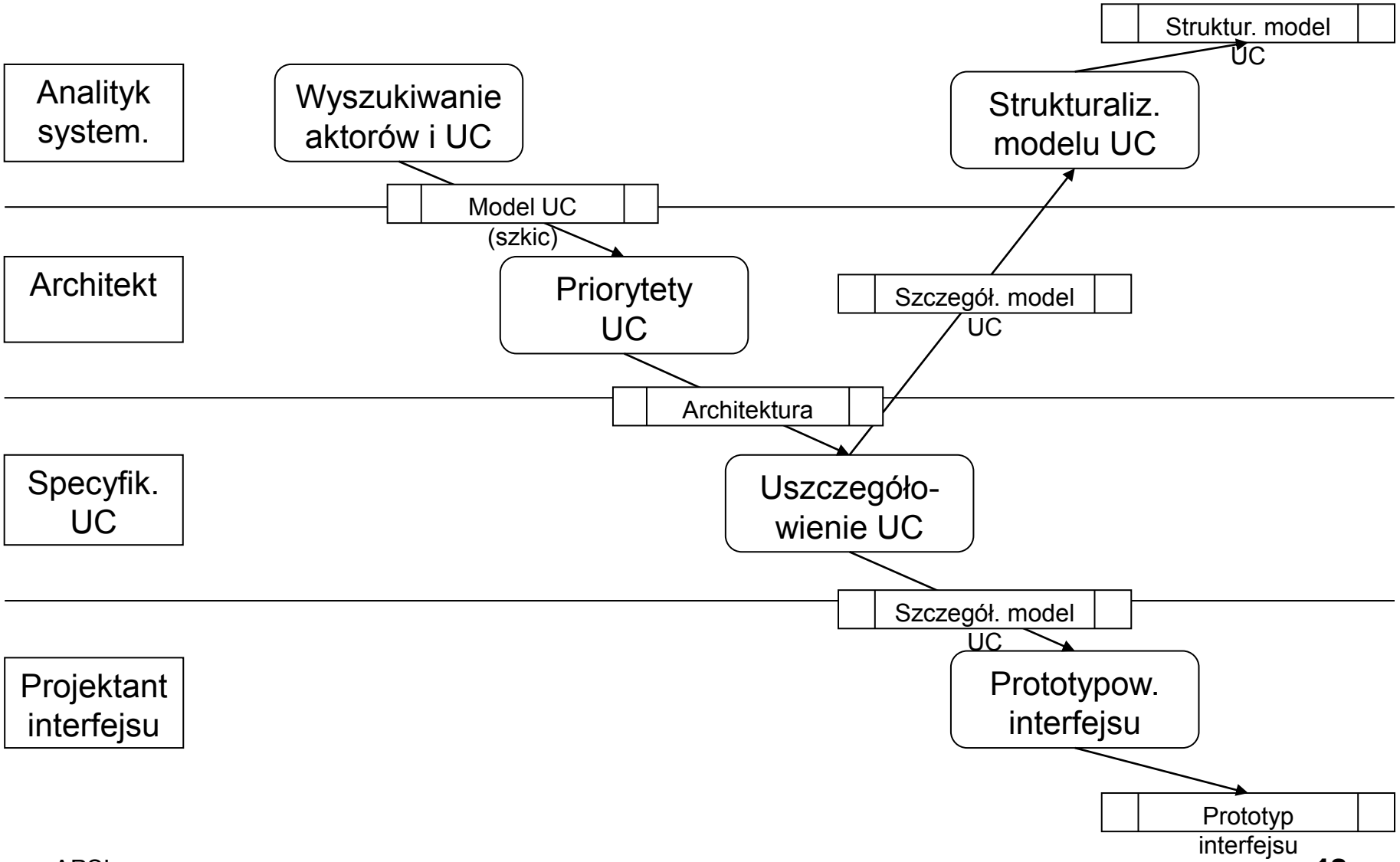
Analiza przypadków użycia

- rozszerzanie i uszczegóławianie opisów
- wyszukiwanie związków pomiędzy przypadkami użycia
  - wyodrębnianie zachowań wspólnych (include, extends)
  - ustalanie szablonów interakcji (generalizacja)
- weryfikacja jakości modelu przypadków użycia:
  - niesprzeczność,
  - spójność,
  - kompletność

Zarys procesu modelowania

- identyfikacja aktorów i przypadków użycia,
- zapis przypadków użycia w formacie wysokiego poziomu,
- określenie wagi i rankingu aktorów i przypadków użycia,
- analiza – zapis przypadków użycia w formie rozszerzonej,
- strukturalizacja i powiązanie przypadków użycia,
- projektowanie – przechodzenie od przypadków typu istotnego *(essential)* do rzeczywistego *(real)*.

Diagram procesu



Historyjka użytkownika (*User Story*)

Historyjki użytkowników są techniką wykorzystywaną w zwinnych metodach wytwarzania oprogramowania jako narzędzie opisu funkcjonalności systemu.

Technika została wprowadzona w XP, a rozwinięta w metodzie Scrum.

Historyjka użytkownika to krótkie określenie intencji opisujące, czego użytkownik oczekuje od systemu:

Jako <*rola*>

mogę <*działanie*>

aby <*korzyść*>.

Historyjki użytkownika – cechy

- zorientowane na cel użytkowy, nie cechy produktu
- krótkie, możliwe do realizacji w jednej iteracji
- zrozumiałe dla wszystkich
- łatwe do oszacowania pracochłonności, użyteczne do planowania
- zapewniające możliwość odłożenia specyfikacji szczegółów do najlepszego momentu

Historyjki użytkownika – cechy

Uwaga! Historyjki **nie są** specyfikacją wymagań!

W podejściu zwinnym nie koncentrujemy się na definiowaniu szczegółowej specyfikacji (która nigdy nie będzie dość szczegółowa i precyzyjna), ale na maksymalnym ułatwieniu rozwoju oprogramowania.

Historyjki użytkownika należy traktować raczej jako narzędzie organizacji pracy, odniesienie do specyfikacji, niż samą specyfikację.

Historyjki użytkownika – cechy

INVEST:

- **I**ndependent
- **N**egotiable
- **V**erifiable
- **E**stimable
- **S**mall
- **T**estable

Independent

Każda historyjka powinna być maksymalnie niezależna od innych, jeżeli chodzi o:

- szacowanie
- implementację
- testowanie
- wdrożenie

Negotiable

Historyjki nie stanowią specyfikacji wymagań, nie są więc „kontraktem” pomiędzy klientem i wykonawcą.

Dyskusja nad każdą historyjką jest jej integralną częścią. Pozwala to ułatwić komunikację i przyspieszyć realizację.

Negocjowalność daje elastyczność – pozwala ustalić równowagę pomiędzy rozbudowaną funkcjonalnością a kosztem i czasem realizacji.

Valuable

Każda historyjka powinna zapewniać konkretną wartość użytkową. Dlatego należy pisać je z perspektywy tego uczestnika, który rzeczywiście osiąga korzyść (dla którego stanowi wartość).

Należy przyjąć zasadę wertykalnego podziału systemu, dzięki czemu każdy wytworzony inkrement będzie mógł realizować konkretną funkcjonalność użytkową.

Estimable

Dla każdej historyjka powinno dać się oszacować jej złożoność i pracochłonność. Konieczne jest co najmniej stwierdzenie, czy da się ja zrealizować w ramach jednej iteracji.

Dyskusja dotycząca estymacji sama w sobie stanowi wartość, ponieważ pozwala lepiej zrozumieć istotę problemu, odkryć ewentualne ukryte założenia czy pominięte kryteria akceptacji.

Plan jest niczym, planowanie – wszystkim.

D. Eisenhower (?)

Small

Historyjki powinny być na tyle „małe”, aby każda dała się zrealizować w jednej iteracji.

Mniejsze jednostki pracy dają większą elastyczność, co prowadzi do zwiększonej wydajności.

Mniejsza złożoność upraszcza implementację. Należy zatem dekomponować duże historie, tzw. epopeje *(epics)* na mniejsze. Jest to naturalny proces definiowania historyjek – odkrywanie nowych szczegółów i warunków pozwala różnicować i wyodrębniać mniejsze jednostki.

Zbiór powiązanych historyjek tworzy „temat”. W ramach tematu można obserwować zależności.

Dekompozycji nie poddają się problemy skomplikowane.

Small - dekompozycja

- podział według kroków procesu
- podział według różnorodności danych
- podział według różnic scenariuszy
- podział według różnych przypadków reguł biznesowych
- podział według interfejsu / metod dostępu
- wyodrębnienie najbardziej pracochłonnego przypadku
- ograniczanie złożoności i zwiększanie jej później
- opóźnienie realizacji cech jakościowych (upraszczanie)
- podział według operacji (C/R/U/D)

Testable

Każda historyjka musi mieć kryteria akceptacji.

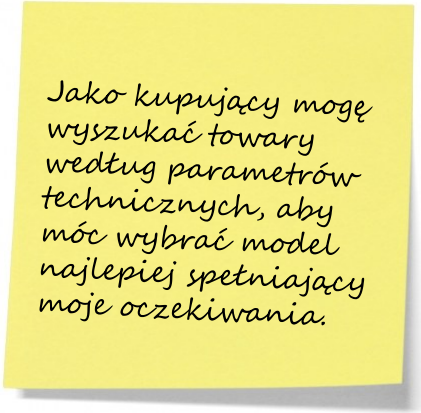
W trakcie dyskusji konieczne jest ujednoznacznienie i uszczegółowienie wszelkich sformułowań niejednoznacznych lub nieścisłych.

Na podstawie kryteriów akceptacji tworzone są testy. Powinny one powstawać przed lub równolegle z oprogramowaniem.

Historyjki użytkownika – forma

CCC:

- **Card**: krótki opis istoty historyjki (zapisywany tradycyjnie na karteczkach)
- **Conversation**: rozmowy pomiędzy członkami zespołu, klientem, właścicielem produktu, etc. prowadzące do dookreślenia wszystkich istotnych aspektów użytkowych (zaproszenie do dyskusji)
- **Confirmation**: definicja testów akceptacyjnych pozwalających ocenić i potwierdzić, czy cel i wszystkie wymagania użytkowe zostały spełnione



Historyjki użytkownika – szczegóły

Rezultaty dyskusji, wyjaśnienia, dodatkowe założenia, wymagania, szczegóły, kryteria akceptacji (warunki testów) stanowią załączniki do historyjki. Są one zapisywane i przechowywane w odpowiedniej dla nich formie.

Uwaga! Kryteria akceptacji **nie są** specyfikacją testów. Są to jedynie warunki prowadzące do uzyskania satysfakcji ze strony użytkownika.

Historyjki a przypadki użycia

Epopeja często odpowiada zakresem przypadkowi użycia.

Ostateczna historyjka użytkownika po dekompozycji najczęściej odpowiada jednemu z przebiegów w ramach przypadku użycia z dołączonym odpowiednim zestawem kryteriów akceptacji.

Poziom opisu historyjek odpowiada istotnym przypadkom użycia.

Odpowiednikiem rzeczywistych przypadków użycia są testy.

Obie techniki mogą być stosowane razem, ale nie w projekcie prowadzonym metodą tradycyjną.

Historyjki a przypadki użycia

Use Cases

- stanowią specyfikację
- są dokumentacją
- są akceptowane („kontrakt”)
- są trwałe
- pełny zakres funkcjonalności
- pełny opis
- pełny obraz całości
- zawierają szczegóły implementacyjne (rzeczywiste)
- nie nadają się do planowania

User Stories

- nie są specyfikacją
- ułatwiają organizację
- służą dyskusji
- są ulotne
- wąski zakres
- zarys opisu
- wąska perspektywa
- są pozbawione szczegółów implementacyjnych
- służą planowaniu

Wymagania niefunkcjonalne

- ilościowe
  - określone ilościowo
  - podlegające pomiarom
- jakościowe
  - określone w formie warunków
  - podlegające badaniom

Zasady definiowania wymagań ilościowych

- określenie mierzonych wartości
  - formułowanie założeń w jednostkach biznesowych, a nie fizycznych
    - np. pomiar liczby dokumentów (a nie GB)
  - wyeliminowanie ewentualnych niejednoznaczności w określeniu tego, co ma podlegać pomiarowi
    - np. czas wykonania transakcji na serwerze bazy danych (a nie na stacji klienckiej)
- określenie metody pomiaru
  - zdefiniowanie mechanizmu technologicznego i sposobu (metodologii) wykonywania pomiaru
    - pomiar ręczny / elektroniczny
    - wykorzystanie zewnętrznych narzędzi pomiarowych / wbudowanie mechanizmów w rozwiązanie
    - sposób generowania obciążenia

Zasady definiowania wymagań ilościowych

- określenie środowiska i warunków pomiaru
  - zdefiniowanie sprzętu, oprogramowania systemowego i podstawowego
    - platforma sprzętowo-systemowa, serwery aplikacyjne, bazy danych, technologia sieci, etc.
  - określenie warunków technicznych
    - uruchomione i działające równolegle oprogramowanie, założone obciążenie testowanego systemu i innego oprogramowania
  - określenie warunków organizacyjnych
    - organizacja testów, zespołu testerów
- określenie interpretacji wyników
  - zdefiniowanie sposobu wyliczania rezultatu testu z wyników pomiarów oraz poziomu akceptowalności rezultatu
    - zalecane: N% wyników jest poniżej/powyżej zadanej wartości

Wymagania niefunkcjonalne

- niezawodność, dostępność *(fault-tolerance, accessibility)*
  - dostępność
    - 99,72 – 1 dzień/rok
    - 99,93 – 6 h/rok
    - 99,99 – 1 h/rok
  - MTBF *(Mean Time Between Failures)*
    - maksymalizacja odporności na awarie – redundancja sprzętu, rozwiązania programowe
  - MTTR *(Mean Time To Recover)*
    - minimalizacja czasu odtwarzania – dobór technologii oprogramowania systemowego, procedury administratorskie

Wymagania niefunkcjonalne

- bezpieczeństwo *(security)*
  - klasy zakresu zagrożeń:
    - klasa *Internal Business* – dostęp wyłącznie dla jednostek wewnętrznych organizacji
    - klasa *Contract Business* – dostęp dla podmiotów zewnętrznych związanych umowami
    - klasa *Public Business* – dostęp publiczny
  - najistotniejsze zagrożenia:
    - anonimowy dostęp
    - nieautoryzowany dostęp do danych (odczyt, wstawienie, aktualizacja, usunięcie): w spoczynku / przy przesyłaniu
    - nieautoryzowana instalacja / reinstalacja / usunięcie oprogramowania aplikacyjnego / systemowego
    - uszkodzenie / zniszczenie fizyczne sprzętu / nośników danych

Wymagania niefunkcjonalne

- bezpieczeństwo *(security)*
  - przeciwdziałanie
    - środki techniczne
    - mechanizmy organizacyjne
    - zabezpieczenia fizyczne
  - wykrywanie i działania naprawcze
    - środki techniczne
    - procedury awaryjne

Wymagania niefunkcjonalne

- pojemność *(capacity)*
  - miary ilościowe: liczba użytkowników, elementów systemu, ilość danych
    - dobór technologii pod względem pojemności
- wydajność, sprawność, efektywność *(performance, efficiency, effectiveness)*
  - wydajność: bezwzględna miara ilościowa realizacji funkcji systemu
  - sprawność: miara ilościowa realizacji funkcji systemu względem używanych zasobów
  - efektywność: miara stopnia, w jaki system wspomaga użytkowników w wykonywaniu ich pracy
  - dobór technologii pod względem wydajnościowym

Wymagania niefunkcjonalne

- zarządzalność, łatwość utrzymania *(manageability, maintainability)*
  - łatwość wykonywania procedur administratorskich w ramach rozwiązania, takich jak np. zarządzanie oprogramowaniem, użytkownikami, danymi, monitorowanie, strojenie
    - dobór architektury systemu
    - dobór specjalistycznego oprogramowania
    - wbudowanie mechanizmów w system

Wymagania niefunkcjonalne

- wiarygodność *(reliability)*
  - miara stopnia, w jakim system wykonuje swoje funkcje w sposób poprawny (stopnia zaufania do systemu)
    - dobór technologii
    - jakość i pełność testów
- trwałość, odporność *(robustness)*
  - zdolność systemu do działania pomimo występowania zdarzeń niekorzystnych, awarii
    - dobór technologii odpornych na awarie
    - modularna architektura systemu
    - architektura rozproszona

Wymagania niefunkcjonalne

- użyteczność *(usability)*
  - stopień spełnienia wymagań użytkowników i przynoszenia im korzyści w realizacji ich zadań
- ergonomia *(ergonomics)*
  - łatwość wykorzystywania funkcji systemu
- zrozumiałość *(understandability)*
  - zdolność systemu do komunikowania się z użytkownikami w prosty i odpowiedni dla nich sposób
- wielojęzyczność *(internationalization)*
  - zdolność systemu do komunikowania się w wielu językach

Wymagania niefunkcjonalne

- zgodność *(conformability)*: normy, reguły, infrastruktura
  - zgodność rozwiązania z obowiązującym prawem i innymi regulacjami
- kompatybilność *(compatibility)*
  - zdolność rozwiązania do współpracy z innymi systemami
- organizacja *(organization)*
  - wzajemne dostosowanie systemu i organizacji, w której ma funkcjonować
- topologia *(topology)*
  - sposób konstrukcji rozwiązania pod względem topologii węzłów i sieci w systemie

  - Wszystkie powyższe można rozumieć jako ograniczenia projektu

Wymagania niefunkcjonalne

- modyfikowalność *(modifiability)*
  - zdolność rozwiązania do wnoszenia modyfikacji do jego funkcjonalności
- indywidualizacja *(customizability)*
  - zdolność rozwiązania do dostosowywania interfejsu i funkcjonalności systemu do specyficznych wymagań poszczególnych użytkowników
- uniwersalność, elastyczność *(versatility)*
  - możliwość zastosowania rozwiązań w innych systemach
- przenośność *(portability)*
  - zdolność rozwiązania do działania na różnych platformach sprzętowo-systemowych

Wymagania niefunkcjonalne

- reużywalność *(reusability)*
  - możliwość wykorzystania elementów rozwiązania w wielu elementach danego systemu i/lub w innych systemach
- rozszerzalność *(extendibility)*
  - zdolność systemu do dodawania nowych elementów funkcjonalnych
- skalowalność *(scalability)*
  - zdolność systemu do zwiększania parametrów ilościowych – wydajności, pojemności
  - wszystkie wymienione wymagają:
    - doboru odpowiedniej technologii
    - opracowania odpowiedniej architektury

Wymagania niefunkcjonalne

- kompletność *(completeness)*
  - miara stopnia pokrycia przez system funkcjonalności istotnej dla organizacji i użytkowników
    - pełność testów i zapewnienie zgodności z modelem przypadków użycia
- testowalność *(testability)*
  - zdolność rozwiązania do łatwego, wygodnego i efektywnego testowania
    - wykorzystanie narzędzi do automatyzacji procesu testowania
    - wbudowanie mechanizmów w system