Table of Contents

Walka z kryzysem oprogramowania

- Stosowanie technik i narzędzi ułatwiających pracę nad złożonymi systemami;
- Korzystanie z metod wspomagających analizę nieznanych problemów oraz ułatwiających wykorzystanie wcześniejszych doświadczeń;
- Usystematyzowanie procesu wytwarzania oprogramowania, tak aby ułatwić jego planowanie i monitorowanie;
- Wytworzenie wśród producentów i nabywców przekonania, że budowa dużego systemu wysokiej jakości jest zadaniem wymagającym profesjonalnego podejścia.

Podstawowym powodem kryzysu oprogramowania jest
złożoność produktów informatyki i procesów ich wytwarzania

Źródła złożoności projektu oprogramowania

- **Dziedzina problemowa** obejmująca ogromną liczbę wzajemnie uzależnionych aspektów i problemów.
- **Środki i technologie informatyczne:** sprzęt, oprogramowanie, sieć, języki, narzędzia, udogodnienia.
- **Zespół projektantów** podlegający ograniczeniom pamięci, percepcji, wyrażania informacji i komunikacji.
- **Potencjalni użytkownicy**: czynniki psychologiczne, ergonomia, ograniczenia pamięci i percepcji, skłonność do błędów i nadużyć, tajność, prywatność.

Jak walczyć ze złożonością

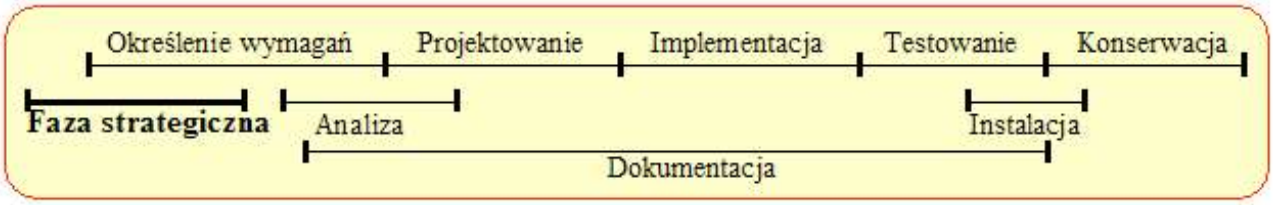
- ***Zasada dekompozycji:*** rozdzielenie złożonego problemu na podproblemy, które można rozpatrywać i rozwiązywać niezależnie od siebie i niezależnie od całości.
- ***Zasada abstrakcji:*** eliminacja, ukrycie lub pominięcie mniej istotnych szczegółów rozważanego przedmiotu lub mniej istotnej informacji; wyodrębnianie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzaniu pojęć lub symboli oznaczających takie cechy.
- ***Zasada ponownego użycia*****:** wykorzystanie wcześniej wytworzonych schematów, metod, wzorców, komponentów projektu, komponentów oprogramowania, itd.
- ***Zasada sprzyjania naturalnym ludzkim własnościom*****:** dopasowanie modeli pojęciowych i

modeli realizacyjnych systemów do wrodzonych ludzkich własności psychologicznych, instynktów oraz mentalnych mechanizmów percepcji i rozumienia świata.

Metodyka jest to zestaw pojęć, notacji, modeli, języków, technik i sposobów postępowania służący do analizy dziedziny stanowiącej przedmiot projektowanego systemu oraz do projektowania pojęciowego, logicznego i/lub fizycznego.

Metodyka jest powiązana z **notacją** służącą do dokumentowania wyników faz projektu (pośrednich, końcowych), jako środek wspomagający ludzką pamięć i wyobraźnię i jako środek komunikacji w zespołach oraz pomiędzy projektantami i klientem.

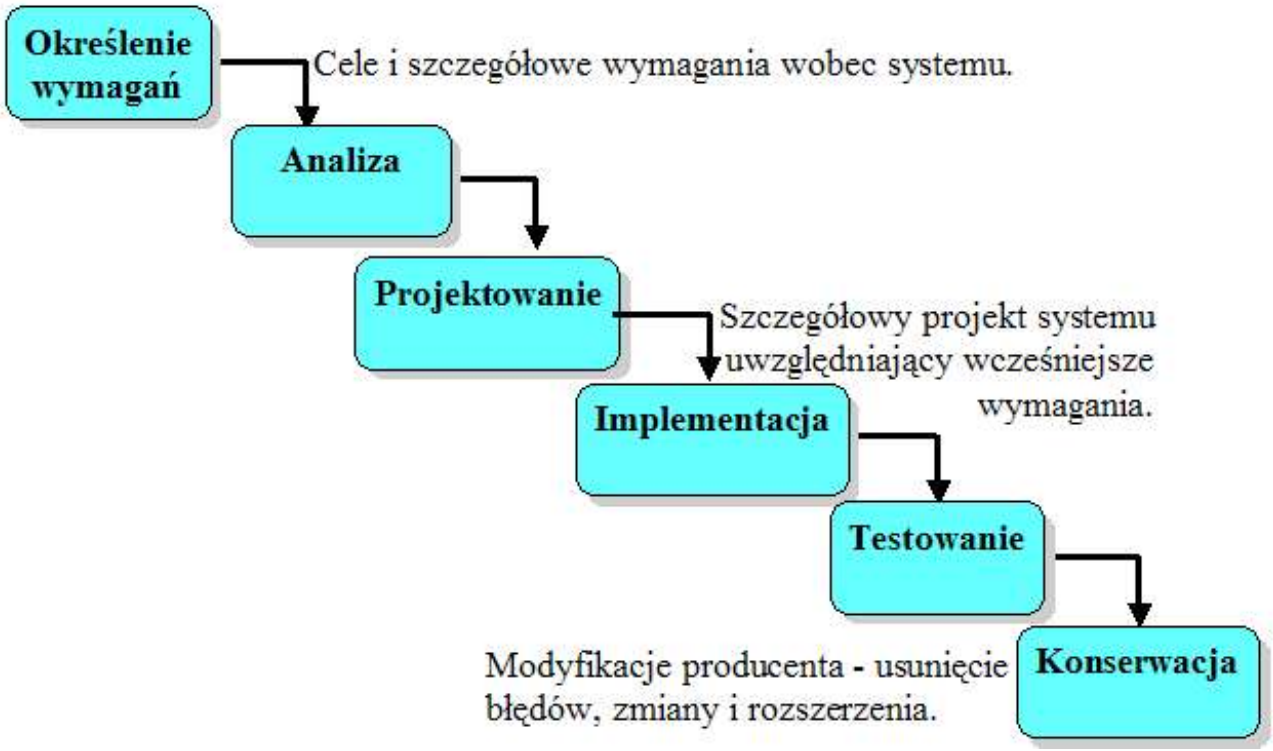
Cykl życia oprogramowania



- Faza strategiczna: określenie strategicznych celów, planowanie i definicja projektu
- Określenie wymagań
- Analiza: dziedziny przedsiębiorczości, wymagań systemowych
- Projektowanie: projektowanie pojęciowe, projektowanie logiczne
- Implementacja/konstrukcja: rozwijanie, testowanie, dokumentacja
- Testowanie
- Dokumentacja
- Instalacja
- Przygotowanie użytkowników, akceptacja, szkolenie
- Działanie, włączając wspomaganie tworzenia aplikacji
- Utrzymanie, konserwacja, pielęgnacja

Model kaskadowy

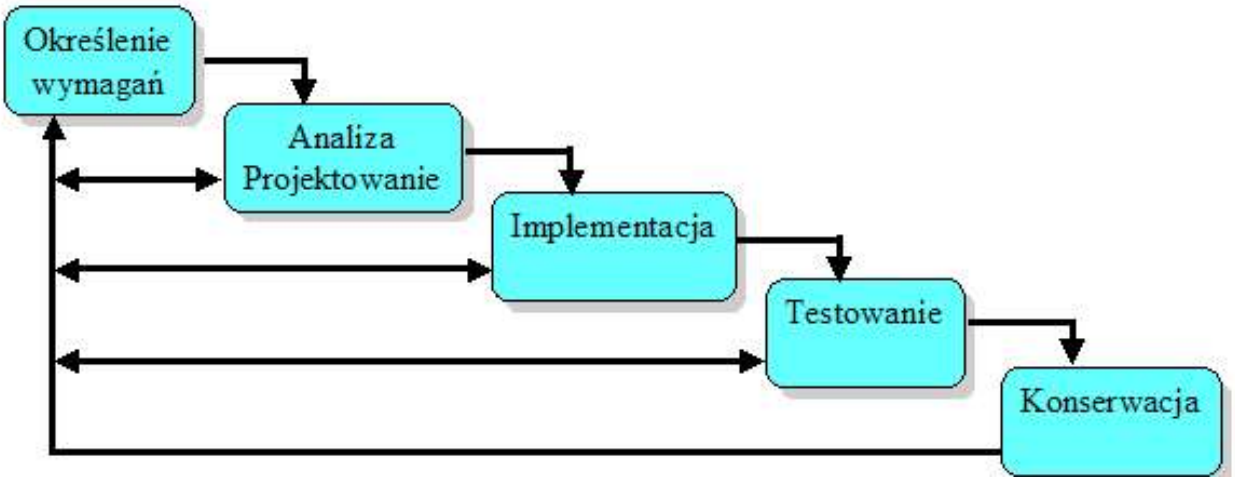
Wady modelu kaskadowego:



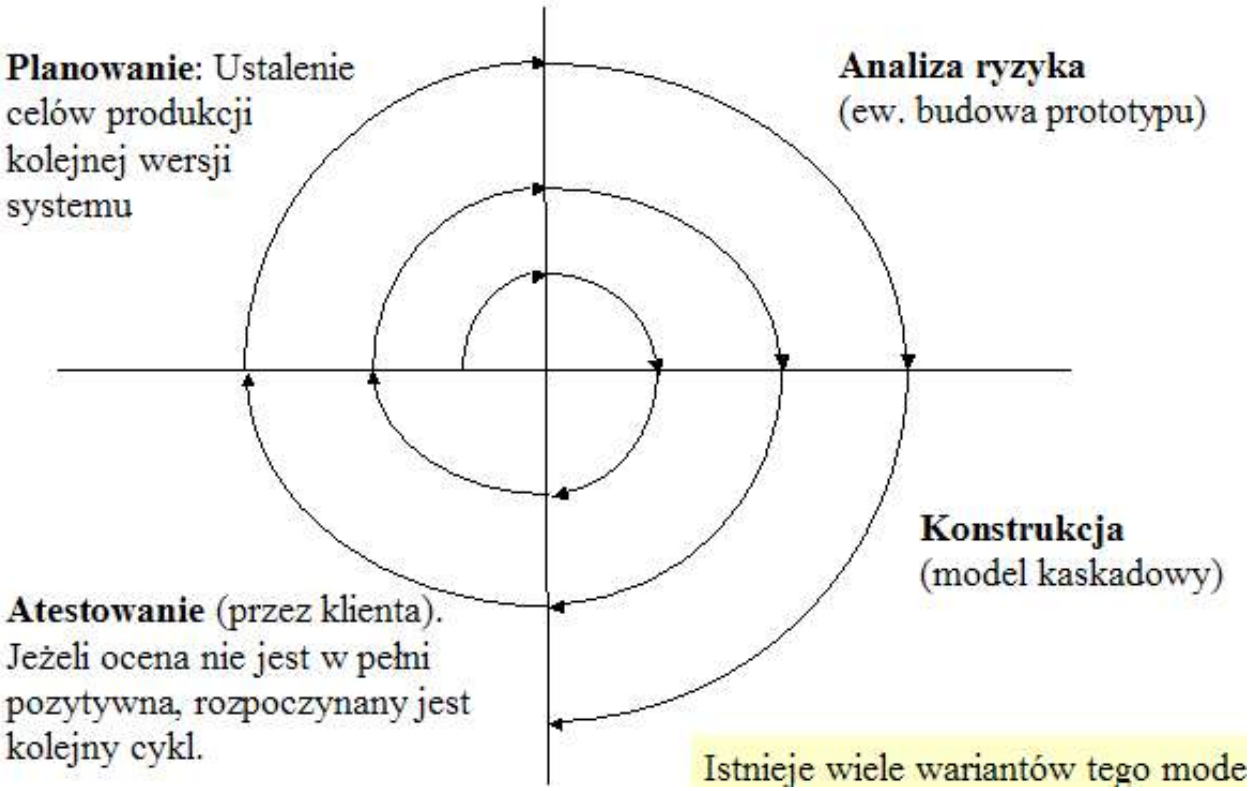
- Narzucenie twórcom oprogramowania ścisłej kolejności wykonywania prac
- Wysoki koszt błędów popełnionych we wczesnych fazach
- Długa przerwa w kontaktach z klientem

Z drugiej strony, jest on do pewnego stopnia niezbędny dla planowania, harmonogramowania, monitorowania i rozliczeń finansowych.

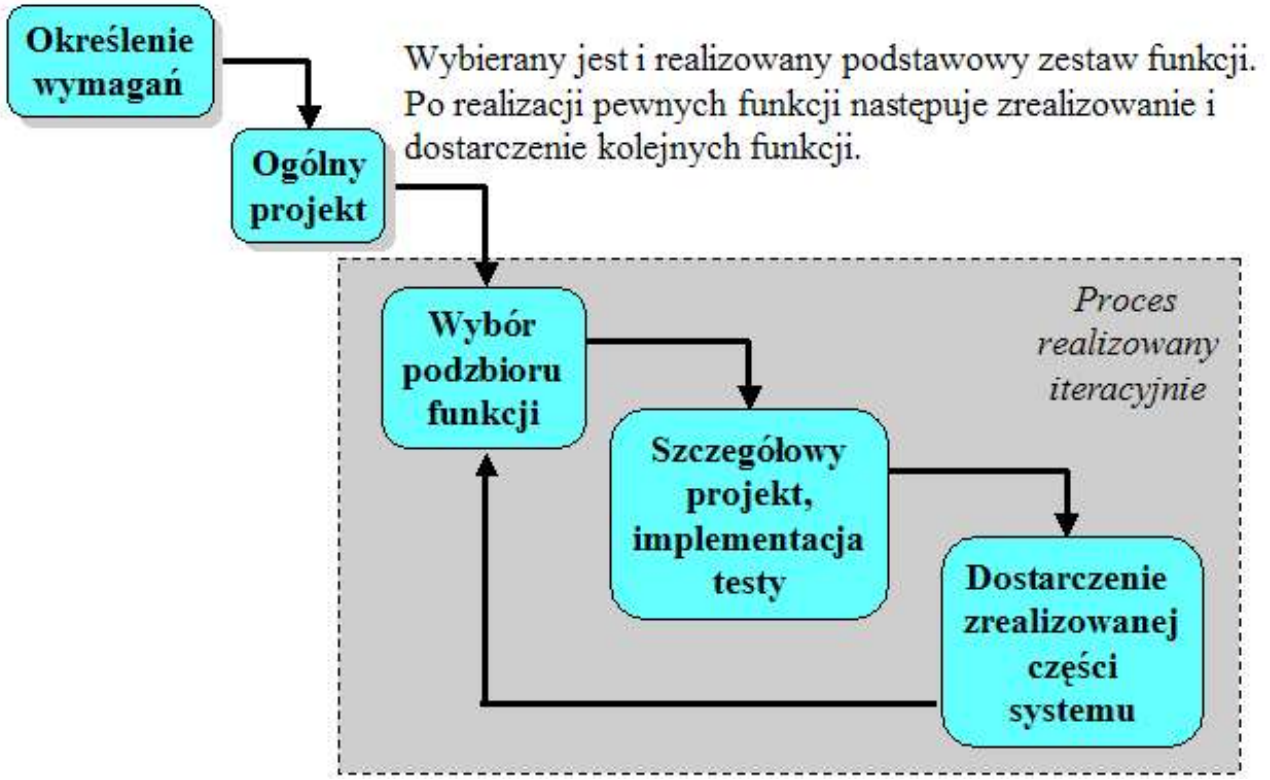
Model kaskadowy z iteracjami



Model spiralny



Realizacja przyrostowa



Prototypowanie

Fazy

- ogólne określenie wymagań

- budowa prototypu
- weryfikacja prototypu przez klienta
- pełne określenie wymagań
- realizacja pełnego systemu zgodnie z modelem kaskadowym

Cele

- wykrycie nieporozumień pomiędzy klientem a twórcami systemu
- wykrycie brakujących funkcji
- wykrycie trudnych usług
- wykrycie braków w specyfikacji wymagań

Zalety

- możliwość demonstracji pracującej wersji systemu
- możliwość szkoleń zanim zbudowany zostanie pełny system

Montaż z gotowych komponentów

Metody

- zakup elementów ponownego użycia od dostawców
- przygotowanie elementów poprzednich przedsięwzięć do ponownego użycia

Zalety

- wysoka niezawodność
- zmniejszenie ryzyka
- efektywne wykorzystanie specjalistów
- narzucenie standardów

Wady

- dodatkowy koszt przygotowania elementów ponownego użycia
- ryzyko uzależnienia się od dostawcy elementów
- niedostatki narzędzi wspomagających ten rodzaj pracy

Faza strategiczna

- Cel – ogólne informacje co ma być osiągnięte dzięki systemowi
- Zakres – określenie fragmentu procesów informacyjnych zachodzących w organizacji, które będą objęte przedsięwzięciem. Na tym etapie może nie być jasne, które funkcje będą wykonywane przez oprogramowanie, a które przez personel, inne systemy lub standardowe wyposażenie sprzętu. (Czyli chodzi o to co będzie zadaniem systemu)
- Kontekst – systemy, organizacje, użytkownicy zewnętrzni, z którymi tworzony system ma współpracować
- Wymagania funkcjonalne – lista funkcji jakie mają być wykonywane przez system
- Wymagania niefunkcjonalne (ograniczenia) – informacje na temat systemu które nie są związane bezpośrednio z funkcjami realizowanymi przez system

Wymagania niefunkcjonalne

- **Wymagania dotyczące produktu**. Np. musi istnieć możliwość operowania z systemem wyłącznie za pomocą klawiatury.
- **Wymagania dotyczące procesu**. Np. proces realizacji harmonogramowania zleceń musi być zgodny ze standardem opisanym w dokumencie XXXA/96.

- **Wymagania zewnętrzne**. Np. system harmonogramowania musi współpracować z bazą danych systemu komputerowego działu marketingu opisaną w dokumencie YYYB/95. Niedopuszczalne są jakiekolwiek zmiany w strukturze tej bazy.

Wymagania niefunkcjonalne powinny być **weryfikowalne -** czyli powinna istnieć możliwość sprawdzenia lub zmierzenia czy system je rzeczywiście spełnia. Np. wymagania "*system ma być łatwy w obsłudze*", „*system ma być niezawodny*", „*system ma być dostatecznie szybki*", itd. nie są weryfikowalne.

Cechy weryfikowalne

Wydajność
- Liczba transakcji obsłużonych w ciągu sekundy
- Czas odpowiedzi

rozmiar
- Liczba rekordów w bazie danych
- Wymagana pamięć dyskowa

łatwość użytkowania
- Czas niezbędny dla przeszkolenia użytkowników
- Rozmiar dokumentacji

niezawodność
- Prawdopodobieństwo błędu podczas realizacji transakcji
- Średni czas pomiędzy błędnymi wykonaniami
- Dostępność (procent czasu w którym system jest dostępny)
- Czas restartu po awarii systemu
- Prawdopodobieństwo zniszczenia danych w przypadku awarii

przenaszalność
- Procent kodu zależnego od platformy docelowej
- Liczba platform docelowych
- Koszt przeniesienia na nową platformę

Czynniki uwzględniane przy konstruowaniu wymagań niefunkcjonalnych

- Możliwości systemu
- Objętość
- Szybkość
- Dokładność
- Ograniczenia
- Interfejsy komunikacyjne
- Interfejsy sprzętowe
- Interfejsy oprogramowania
- Interakcja człowiek-maszyna
- Adaptowalność
- Bezpieczeństwo
- Odporność na awarie
- Standardy
- Zasoby
- Skala czasowa

Porządkowanie wymagań

Z reguły funkcje można rozbić na podfunkcje.
W notacji graficznej:
- Na każdym poziomie ten sam poziom szczegółowości.

- Kolejność funkcji nie ma znaczenia.
- Niektóre funkcje mogą być wykonywane wielokrotnie.
- Wyszczególniać tylko funkcje widoczne dla użytkowników.

Przykład: Program podatkowy

Cele:
- przyśpieszenie obsługi klientów
- zmniejszenie ryzyka popełnienia błędów

Zakres
działalność jednej firmy rachunkowej, która może mieć dowolną liczbę klientów. Nie jest określone, czy system ma drukować wypełniony PIT, czy tylko dostarczać dane.
Kontekst:
Pracownik firmy

Przykład: system harmonogramowania zleceń

Cele
- zwiększenie wydajności pracy wydziału poprzez szybszą i efektywniejszą realizację zleceń,
- zmniejszenie opóźnień w realizowaniu zleceń
- uwzględnienie wszelkich ograniczeń, zapewniające praktyczną wykonalność proponowanych harmonogramów
- zapewnienie możliwości "ręcznego" modyfikowania harmonogramu
- opracowanie harmonogramu w formie łatwej do wykorzystania przez kadrę kierowniczą wydziału oraz automatyzacja przygotowania zamówień dla magazynu na półprodukty.

Zakres
funkcjonowanie komórki wydziału obejmującego przygotowanie harmonogramu wykonywania zleceń
Kontekst
system komputerowy działu marketingu, osoba definiująca technologiczne możliwości wydziału, kadra kierownicza.

Przykłady ograniczeń

- Maksymalne nakłady, jakie można ponieść na realizację przedsięwzięcia
- Dostępny personel
- Dostępne narzędzia
- Ograniczenia czasowe

Decyzje strategiczne

- Wybór modelu, zgodnie z którym będzie realizowane przedsięwzięcie
- Wybór technik stosowanych w fazach analizy i projektowania
- Wybór środowiska (środowisk) implementacji
- Wybór narzędzia CASE
- Określenie stopnia wykorzystania gotowych komponentów
- Podjęcie decyzji o współpracy z innymi producentami lub zatrudnieniu ekspertów

Szacowanie złożoności oprogramowania

Metoda COCOMO

(COnstructive COst MOdel) – szacowanie złożoności na podstawie oszacowanej ilości liczby linii kodu
COCOMO oferuje kilka metod określanych jako podstawowa, pośrednia i detaliczna.

- **Metoda podstawowa:** prosta formuła dla oceny osobo-miesięcy oraz czasu potrzebnego na całość projektu.
- **Metoda pośrednia**: modyfikuje wyniki osiągnięte przez metodę podstawową poprzez odpowiednie czynniki, które zależą od aspektów złożoności.
- **Metoda detaliczna**: bardziej skomplikowana, ale jak się okazało, nie dostarcza lepszych wyników niż metoda pośrednia.

Metoda punktów funkcyjnych

Empiryczna metoda oceny złożoności realizacji projektów informatycznych poprzez tzw. punkty funkcyjne (function points, FP). Polega na wydzieleniu atrybutów produktywności (miar pracy) w projektach informatycznych. Na podstawie szacowanych wartości atrybutów produktywności dla danego projektu ocenia się ilość punktów funkcyjnych jako miarę produktywności zespołu lub złożoności projektu. Atrybutami produktywności projektowanego systemu informatycznego są: wejścia użytkownika (dane, sterowanie), wyjścia użytkownika (wydruki, ekrany, pliki), zbiory danych wewnętrzne, zbiory danych zewnętrzne, zapytania zewnętrzne. Szacunkowe oceny są poddawane korekcji uwzględniającej warunki realizacji systemu informatycznego.

Elementy	Proste	Waga	Średnie	Waga	Złożone	Waga	Razem
Wejścia	2 →	3	5 →	4	3 →	6	44
Wyjścia	10 →	4	4 →	5	5 →	7	95
Zbiory wew.	3 →	7	5 →	10	2 →	15	101
Zbiory zew.	0 →	5	3 →	7	0 →	10	21
Zapytania	10 →	3	5 →	4	12 →	6	122
							Łącznie 383

Czynniki modyfikacji punktów funkcyjnych

- Występowanie urządzeń komunikacyjnych
- Rozproszenie przetwarzania
- Wymagane parametry szybkości działania
- Skomplikowana logika przetwarzania
- Obciążenie systemu - liczba transakcji
- Wprowadzanie danych w trybie online
- Wydajność użytkownika końcowego
- Aktualizacja danych w trybie online
- Rozproszenie terytorialne
- Złożoność przetwarzania danych
- Przenośność

- Prostota instalacji
- Prostota obsługi
- Zmiany w okresie eksploatacji

Przeliczanie punktów funkcyjnych

- 1 FP ≈ 125 instrukcji w C
- 10 FPs - typowy mały program tworzony samodzielnie przez klienta (1 m-c)
- 100 FPs - większość popularnych aplikacji; wartość typowa dla aplikacji tworzonych przez klienta samodzielnie (6 m-cy)
- 1,000 FPs - komercyjne aplikacje w MS Windows, małe aplikacje klient-serwer (10 osób, ponad 12 m-cy)
- 10,000 FPs - systemy (100 osób, ponad 18 m-cy)
- 100,000 FPs - MS Windows'95, MVS, systemy militarne

Dokument Specyfikacji Wymagań

Nagłówek:
Streszczenie (maksymalnie 200 słów)
Spis treści
Status dokumentu (autorzy, firmy, daty, podpisy, itd.)
Zmiany w stosunku do wersji poprzedniej

Spis treści
1. Wstęp
  1.1. Cel
  1.2. Zakres
  1.3. Definicje, akronimy i skróty
  1.4. Referencje, odsyłacze do innych dokumentów
  1.5. Krótki przegląd
2. Ogólny opis
  2.1. Walory użytkowe i przydatność projektowanego systemu
  2.2. Ogólne możliwości projektowanego systemu
  2.3. Ogólne ograniczenia
  2.4. Charakterystyka użytkowników
  2.5. Środowisko operacyjne
  2.6. Założenia i zależności
3. Specyficzne wymagania
  3.1. Wymagania funkcjonalne (funkcje systemu)
  3.2. Wymagania niefunkcjonalne (ograniczenia).
Dodatki

Kolejność i numeracja punktów powinna być zachowana. W przypadku gdy pewien punkt nie zawiera żadnej informacji należy wyraźnie to zasygnalizować przez umieszczenie napisu „Nie dotyczy".

Często spotykane dodatki:
- Wymagania sprzętowe.
- Wymagania dotyczące bazy danych.
- Model (architektura) systemu.
- Słownik terminów (użyte terminy, akronimy i skróty z wyjaśnieniem)
- Indeks pomocny w wyszukiwaniu w dokumencie konkretnych informacji (dla dokumentów dłuższych niż 80 stron)

Słownik

Opis wymagań może zawierać terminy niezrozumiałe dla jednej ze stron. Mogą to być **terminy informatyczne (niezrozumiałe dla klienta)** lub **terminy dotyczące dziedziny zastosowań (niezrozumiałe dla przedstawicieli producenta**).
Wszystkie specyficzne terminy powinny być umieszczone w słowniku, wraz z wyjaśnieniem.
Słownik powinien precyzować terminy niejednoznaczne i określać ich znaczenie w kontekście tego dokument (być może nieco węższe).
Przykład:
Termin: konto
Objaśnienie: Nazwana ograniczona przestrzeń dyskowa, gdzie użytkownik może przechowywać swoje dane. Konta są powiązane z określonymi usługami, np. pocztą komputerową oraz z prawami dostępu.
Termin: konto bankowe
Objaśnienie: Sekwencja cyfr oddzielona myślnikami, identyfikująca stan zasobów finansowych oraz operacji dla pojedynczego klienta banku.

Faza analizy

Czynności:
- Rozpoznanie, wyjaśnianie, modelowanie, specyfikowanie i dokumentowanie rzeczywistości lub problemu będącego przedmiotem projektu;
- Ustalenie kontekstu projektu;
- Ustalenie wymagań użytkowników;
- Ustalenie wymagań organizacyjnych
- Inne ustalenia, np. dotyczące preferencji sprzętowych, preferencji w zakresie oprogramowania, ograniczeń finansowych, ograniczeń czasowych, itd.

Dokument Wymagań na Oprogramowanie

Streszczenie (maksymalnie 200 słów)
Spis treści
Status dokumentu (autorzy, firmy, daty, podpisy, itd.)
Zmiany w stosunku do wersji poprzedniej
1. Wstęp
1.1. Cel
1.2. Zakres
1.3. Definicje, akronimy i skróty
1.4. Referencje, odsyłacze do innych dokumentów
1.5. Krótki przegląd
2. Ogólny opis
2.1. Relacje do bieżących projektów
2.2. Relacje do wcześniejszych i następnych projektów
2.3. Funkcje i cele
2.4. Ustalenia dotyczące środowiska
2.5. Relacje do innych systemów
2.6. Ogólne ograniczenia
2.7. Opis modelu
3. Specyficzne wymagania (ten rozdział może być podzielony na wiele rozdziałów zgodnie z podziałem funkcji)
3.1. Wymagania dotyczące funkcji systemu
3.2. Wymagania dotyczące wydajności systemu
3.3. Wymagania dotyczące zewnętrznych interfejsów

3.4. Wymagania dotyczące wykonywanych operacji
3.5. Wymagania dotyczące wymaganych zasobów
3.6. Wymagania dotyczące sposobów weryfikacji
3.7. Wymagania dotyczące sposobów testowania
3.8. Wymagania dotyczące dokumentacji
3.9. Wymagania dotyczące ochrony
3.10. Wymagania dotyczące przenośności
3.11. Wymagania dotyczące jakości
3.12. Wymagania dotyczące niezawodności
3.13. Wymagania dotyczące pielęgnacyjności
3.14. Wymagania dotyczące bezpieczeństwa

Dodatki (to, co nie zmieściło się w powyższych punktach)

Zasady projektowania UI

Spójność. Wygląd oraz obsługa interfejsu powinna być podobna w momencie korzystania z różnych funkcji. Poszczególne programy tworzące system powinny mieć zbliżony interfejs, podobnie powinna wyglądać praca z rozmaitymi dialogami, podobnie powinny być interpretowane operacje wykonywane przy pomocy myszy. Proste reguły:

- Umieszczanie etykiet zawsze nad lub obok pól edycyjnych
- Umieszczanie typowych pól OK i Anuluj zawsze od dołu lub od prawej.
- Spójne tłumaczenie nazw angielskich, spójne oznaczenia pól.

Skróty dla doświadczonych użytkowników. Możliwość zastąpienia komend w paskach narzędziowych przez kombinację klawiszy.

Potwierdzenie przyjęcia zlecenia użytkownika. Realizacja niektórych zleceń może trwać długo. W takich sytuacjach należy potwierdzić przyjęcie zlecenie, aby użytkownik nie był zdezorientowany odnośnie tego co się dzieje. Dla długich akcji - wykonywanie sporadycznych akcji na ekranie (np. wyświetlanie sekund trwania, sekund do przewidywanego zakończenia, „termometru", itd.).

Prosta obsługa błędów. Jeżeli użytkownik wprowadzi błędne dane, to po sygnale błędu system powinien automatycznie przejść do kontynuowania przez niego pracy z poprzednimi poprawnymi wartościami.

Odwoływanie akcji (*undo*). W najprostszym przypadku jest to możliwość cofnięcia ostatnio wykonanej operacji. Jeszcze lepiej jeżeli system pozwala cofnąć się dowolnie daleko w tył.

Wrażenie kontroli nad systemem. Użytkownicy nie lubią, kiedy system sam robi coś, czego użytkownik nie zainicjował, lub kiedy akcja systemu nie daje się przerwać. System nie powinien inicjować długich akcji (np. składowania) nie informując użytkownika co w tej chwili robi oraz powinien szybko reagować na sygnały przerwania akcji (Esc, Ctrl+C, Break,...)

Nieobciążanie pamięci krótkotrwałej użytkownika. Użytkownik może zapomnieć o tym po co i z jakimi danymi uruchomił dialog. System powinien wyświetlać stale te informacje, które są niezbędne do tego, aby użytkownik wiedział, co aktualnie się dzieje i w którym miejscu interfejsu się znajduje.

Grupowanie powiązanych operacji. Jeżeli zadanie nie da się zamknąć w prostym dialogu lub oknie, wówczas trzeba je rozbić na szereg powiązanych dialogów. Użytkownik powinien być prowadzony przez ten szereg, z możliwością łatwego powrotu do wcześniejszych akcji.

Reguła Millera 7 +/- 2.

Człowiek może się jednocześnie skupić na 5 - 9 elementach.
Ta reguła powinna być uwzględniana przy projektowaniu interfejsu użytkownika. Dotyczy to liczby opcji menu, podmenu, pól w dialogu, itd. Ograniczenie to można przełamać poprzez grupowanie w wyraźnie wydzielone grupy zestawów semantycznie powiązanych ze sobą elementów.

Bazy danych

Zalety

- Wysoka efektywność i stabilność
- Bezpieczeństwo i prywatność danych, spójność i integralność przetwarzania
- Automatyczne sprawdzanie warunków integralności danych
- Wielodostęp, przetwarzanie transakcji
- Rozszerzalność (zarówno dodawanie danych jak i dodawanie ich rodzajów)
- Możliwość geograficznego rozproszenia danych
- Możliwość kaskadowego usuwania powiązanych danych
- Dostęp poprzez języki zapytań (SQL, OQL)

Wady relacyjnych

- Konieczność przeprowadzenie nietrywialnych odwzorowań przy przejściu z modelu pojęciowego (np. w OMT) na strukturę relacyjną.
- Ustalony format krotki powodujący trudności przy polach zmiennej długości.
- Trudności (niesystematyczność) reprezentacji dużych wartości (grafiki, plików tekstowych, itd.)
- W niektórych sytuacjach - duże narzuty na czas przetwarzania
- Niedopasowanie interfejsu dostępu do bazy danych (SQL) do języka programowania (np. C), określana jako “niezgodność impedancji”.
- Brak możliwości rozszerzalności typów (zagnieżdżania danych)
- Brak systematycznego podejścia do informacji proceduralnej (metod)

Optymalizacja projektu

- Zmiana algorytmu przetwarzania
- Wyłowienie “wąskich gardeł”
- Zaprogramowanie “wąskich gardeł” w języku niższego poziomu
- Denormalizacja relacyjnej bazy danych
- Stosowanie indeksów, tablic wskaźników i innych struktur pomocniczych
- Analiza mechanizmów buforowania danych

Poprawność projektu

Poprawny projekt musi być:
- kompletny
- niesprzeczny
- spójny
- zgodny z regułami składniowymi notacji

Kompletność projektu oznacza, że zdefiniowane są:
- wszystkie klasy
- wszystkie pola (atrybuty)
- wszystkie metody
- wszystkie dane złożone i elementarne
- opisany jest sposób realizacji wszystkich wymagań funkcjonalnych.

Spójność projektu oznacza semantyczną zgodność wszystkich informacji zawartych na poszczególnych diagramach i w specyfikacji.

Pod terminem *jakość* rozumie się bardziej szczegółowe kryteria:
- spójność
- stopień powiązania składowych
- przejrzystość

Kryteria podziału projektu

Spójność opisuje na ile poszczególne części projektu pasują do siebie.
Istotne staje się kryterium podziału projektu na części.
W zależności od tego kryterium, możliwe jest wiele rodzajów spójności
- **Podział przypadkowy**. Podział na moduły (części) wynika wyłącznie z tego, że całość jest za duża (utrudnia wydruk, edycję, itd)
- **Podział logiczny**. Poszczególne składowe wykonują podobne funkcje, np. obsługa błędów, wykonywanie podobnych obliczeń.
- **Podział czasowy**. Składowe są uruchamiane w podobnym czasie, np. podczas startu lub zakończenia pracy systemu.
- **Podział proceduralny (sekwencyjny)**. Składowe są kolejno uruchamiane. Dane wyjściowe jednej składowej stanowią wejście innej
- **Podział komunikacyjny**. Składowe działają na tym samym zbiorze danych wejściowych i wspólnie produkują zestaw danych wyjściowych
- **Podział funkcjonalny**. Wszystkie składowe są niezbędne dla realizacji jednej tej samej funkcji.

Przejrzystość

- **Odzwierciedlenie rzeczywistości**. Składowe i ich związki pojawiające się w projekcie powinny odzwierciedlać strukturę problemu. Ścisły związek projektu z rzeczywistością.
- **Spójność oraz stopień powiązania składowych**.
- **Zrozumiałe nazewnictwo.**
- **Czytelna i pełna specyfikacja**
- **Odpowiednia złożoność składowych na danym poziomie abstrakcji.**

Niebezpieczne techniki programistyczne

- Instrukcja goto
- stosowanie liczb ze zmiennym przecinkiem
- wskaźniki i arytmetyka wskaźników
- obliczenia równoległe
- przerwania i wyjątki
- rekurencja
- dynamiczna alokacja pamięci
- Procedury i funkcje, które realizują wyraźnie odmienne zadania w zależności od parametrów lub stanu zewnętrznych zmiennych
- Niewyspecyfikowane, nieoczekiwane efekty uboczne funkcji i procedur
- złożone wyrażenia bez form nawiasowych
- akcje na danych przez wiele procesów

Rodzaje modyfikacji

- **Modyfikacje poprawiające**: polegają na usuwaniu z oprogramowania błędów popełnionych w fazach wymagań, analizy, projektowania i implementacji .
- **Modyfikacje ulepszające**: polegają na poprawie jakości oprogramowania.
- **Modyfikacje dostosowujące**: polegają na dostosowaniu oprogramowania do zmian zachodzących w wymaganiach użytkownika lub w środowisku komputerowym.

Dokument zgłoszenia błędu

Dokument powinien zawierać:
- Nazwę elementu konfiguracji oprogramowania.
- Wersję lub wydanie tego elementu.
- Priorytet problemu w stosunku do innych problemów (priorytet ma dwa wymiary: **krytyczność** - na ile problem jest istotny dla funkcjonowania systemu, oraz **pilność** - maksymalny czas usunięcia problemu).
- Opis problemu.
- Opis środowiska operacyjnego.
- Zalecane rozwiązanie problemu (o ile użytkownik jest je w stanie określić).
  Problemy mogą wynikać z wielu powodów: błędy w oprogramowaniu, brak funkcji, które okazały się istotne, ograniczenia, których nie uwzględniono lub które się pojawiły, zmiany w środowisku systemu. Stąd każdy ZPO powinien być oceniony odnośnie odpowiedzialności za problem (kto ma ponosić koszt).

Zlecenie zmiany w oprogramowaniu

Powinno zawierać:
- Nazwę elementu konfiguracji oprogramowania.
- Wersję lub wydanie tego elementu.
- Wymagane zmiany.
- Priorytet zlecenia (krytyczność, pilność).
- Personel odpowiedzialny.
- Szacunkową datę początku, datę końca i pracochłonność w osobo-dniach.

ZZO powinien zawierać sekcje dotyczące:
- Zmian w dokumentach wymagań (użytkownika, na oprogramowanie)
- Zmian w dokumentach projektowych (sekcja dla każdego dokumentu)
- Zmian w dokumentach dotyczących zarządzania, testowania, zapewniania jakości.

Składowe narzędzi CASE

- Repozytorium danych
- Generatory kodu
- Generatory dokumentacji technicznej
- Moduł inżynierii odwrotej
- Sprzęgi do narzędzi RAD
- Moduł import/eksportu danych
- Moduł kontroli poprawności
- Moduł kontroli jakości
- Edytory diagramów
- Moduł projektowania interfejsu użytkownika
- Moduł pracy sieciowej
- Moduł zarządzania pracą grupową
- Moduł zarządzania konfiguracjami
- Generatory raportów

Ocena narzędzi CASE

- Zakres oferowanych funkcji i ich zgodność z potrzebami firmy
- Koszt
- Niezawodność
- Opinia o producencie i dystrybutorze
- Dostępność na rynku pracy specjalistów znających dany pakiet
- Stopień zintegrowania z przyjętym środowiskiem programistycznym
- Wielośrodowiskowość
- Koszt szkoleń
- Koszt dostosowania sprzętu do wymagań pakietu

Weryfikacja

- Przeglądy, inspekcje, testowanie, sprawdzanie, audytowanie lub inną działalność ustalającą i dokumentującą czy składowe, procesy, usługi lub dokumenty zgadzają się z wyspecyfikowanymi wymaganiami.
- Oceny systemu lub komponentu mające na celu określenie czy produkt w danej fazie rozwoju oprogramowania spełnia warunki zakładane podczas startu tej fazy.

Weryfikacja włącza następujące czynności:

- Przeglądy techniczne oraz inspekcje oprogramowania.
- Sprawdzanie czy wymagania na oprogramowanie są zgodne z wymaganiami użytkownika.
- Sprawdzanie czy komponenty projektu są zgodne z wymaganiami na oprogramowanie.
- Testowanie jednostek oprogramowania (modułów).
- Testowanie integracji oprogramowania, testowanie systemu.
- Testowanie akceptacji systemu przez użytkowników
- Audyt.

Przeglądy oprogramowania

Formalne przeglądy mogą mieć następującą postać:

- **Przegląd techniczny**. Oceniają elementy oprogramowania na zgodność postępu prac z przyjętym planem.
- **Przejście** (*walkthrough*). Wczesna ocena dokumentów, modeli, projektów i kodu. Celem jest zidentyfikowanie defektów i rozważenie możliwych rozwiązań. Wtórnym celem jest szkolenie i rozwiązanie problemów stylistycznych (np. z formą kodu, dokumentacji, interfejsów użytkownika).
- **Audyt**. Przeglądy potwierdzające zgodność oprogramowania z wymaganiami, specyfikacjami, zaleceniami, standardami, procedurami, instrukcjami, kontraktami i licencjami. Obiektywność audytu wymaga, aby był on przeprowadzony przez osoby niezależne od zespołu projektowego.

Inspekcje

Inspekcja to formalna technika oceny, w której wymagania na oprogramowanie, projekt lub kod są szczegółowo badane przez osobę lub grupę osób nie będących autorami, w celu identyfikacji błędów, naruszenia standardów i innych problemów

Cechy

- Sesje są zaplanowane i przygotowane
- Błędy i problemy są notowane
- Wykonywana przez techników dla techników (bez udziału kierownictwa)
- Dane nie są wykorzystywane do oceny pracowników
- Zasoby na inspekcje są gwarantowane
- Błędy są wykorzystywane w poprawie procesu programowego (prewencja)
- Proces inspekcji jest mierzony

- Proces inspekcji jest poprawiany

Korzyści z inspekcji

- Wzrost produktywności od 30% do 100%
- Skrócenie czasu projektu od 10% do 30%
- Skrócenie kosztu i czasu wykonywania testów od 5 do 10 razy (mniej błędów, mniej testów regresyjnych)
- 10 razy mniejsze koszty pielęgnacji (naprawczej)
- Poprawa procesu programowego
- Dostarczanie na czas (bo wcześnie wiemy o problemach)
- Większy komfort pracy (brak presji czasu i nadgodzin)
- Zwiększenie motywacji
  - świadomość, że produkt będzie oceniany (wybór pomiędzy byciem zażenowanym a dumnym)
  - nauka przez znajdowanie błędów
- Mniejsze koszty marketingu („przykrywanie" braku jakości)

Testy

- Analiza kodu
- Testy funkcjonalne poszczególnych modułów systemu
- Test porównawczy wyników otrzymanych z oczekiwanymi
- Testy obciążenia
- Testy odpornościowe
- Test zużycia zasobów przez system
- Testy wydajności
- Testy niezawodności
- Test bezpieczeństwa
- Test ergonomii interfejsu
- Testy funkcjonalne uruchamiania i działania procesów
- Testy modyfikowalności oprogramowania
- Testy skalowalności systemu
- Testy alfa/beta (alfa jeśli program na zamówienie, beta jeśli na rynek)
- Przegląd jakości dokumentacji

Harmonogram testów

- Testy modułów
  Testy te zostaną przeprowadzone w fazie implementacji, po zakończeniu realizacji poszczególnych modułów.
- Testy systemu
  Testy przeprowadzane po integracji modułów. System jest już testowany jako spójna całość.
- Testy akceptacji (testy alfa)
  Test przeprowadzany przez końcowych użytkowników systemu.

Schemat testów statystycznych

- Losowa konstrukcja danych wejściowych zgodnie z rozkładem prawdopodobieństwa tych danych
- Określenie wyników poprawnego działania systemu na tych danych
- Uruchomienie systemu oraz porównanie wyników jego działania z poprawnymi wynikami.

Testy białej skrzynki

Testowanie n/z białej skrzynki pozwala sprawdzić wewnętrzną logikę programów poprzez odpowiedni dobór danych wejściowych, dzięki czemu można prześledzić wszystkie ścieżki przebiegu sterowania programu

Testy czarnej skrzynki

Tak określa się sprawdzanie funkcji oprogramowania bez zaglądania do środka programu. Testujący traktuje sprawdzany moduł jak „czarną skrzynkę", której wnętrze jest niewidoczne.
Testowanie n/z czarnej skrzynki powinno obejmować cały zakres danych wejściowych.

Klasy równoważności

Testujący powinni podzielić dane wejściowe w „klasy równoważności", co do których istnieje duże przypuszczenie, że będą produkować te same błędy. Np. jeżeli testujemy wartość „Malinowski", to prawdopodobnie w tej samej klasie równoważności jest wartość „Kowalski". Celem jest uniknięcie efektu „eksplozji danych testowych".
„Klasy równoważności" mogą być również zależne od wyników zwracanych przez testowane funkcje. Np. jeżeli wejściem jest wiek pracownika i istnieje funkcja zwracająca wartości „młodociany", „normalny" „wiek emerytalny", wówczas implikuje to odpowiednie klasy równoważności dla danych wejściowych.

Miary niezawodności

- Prawdopodobieństwo błędnego wykonania
- Częstotliwość występowania błędnych wykonań
- Średni czas między błędnymi wykonaniami
- Dostępność

Testy strukturalne

Kryterium pokrycia wszystkich instrukcji. Zgodnie z tym kryterium dane wejściowe należy dobierać tak, aby każda instrukcja została wykonana co najmniej raz. Spełnienie tego kryterium zwykle wymaga niewielkiej liczby testów. To kryterium może być jednak bardzo nieskuteczne.
Kryterium pokrycia instrukcji warunkowych. Dane wejściowe należy dobierać tak, aby każdy elementarny warunek instrukcji warunkowej został co najmniej raz spełniony i co najmniej raz nie spełniony. Testy należy wykonać także dla każdej wartości granicznej takiego warunku.
Testowanie pętli
Kryteria doboru danych wejściowych mogą opierać się o następujące zalecenia:

- Należy dobrać dane wejściowe tak, aby nie została wykonana żadna iteracja pętli, lub, jeżeli to niemożliwe, została wykonana minimalna liczba iteracji.
- Należy dobrać dane wejściowe tak, aby została wykonana maksymalna liczba iteracji.
- Należy dobrać dane wejściowe tak, aby została wykonana przeciętna liczba iteracji.

Typowe błędy wykrywane statycznie

- Niezainicjowane zmienne
- Porównania na równość liczb zmiennoprzecinkowych
- Indeksy wykraczające poza tablice
- Błędne operacje na wskaźnikach
- Błędy w warunkach instrukcji warunkowych
- Niekończące się pętle
- Błędy popełnione dla wartości granicznych (np. > zamiast >=)
- Błędne użycie lub pominięcie nawiasów w złożonych wyrażeniach
- Nieuwzględnienie błędnych danych

Technika posiewania błędów

Polega na tym, że do programu celowo wprowadza się pewną liczbę błędów podobnych do tych, które występują w programie. Wykryciem tych błędów zajmuje się inna grupa programistów niż ta, która dokonała "posiania" błędów.

N oznacza liczbę posianych błędów
M oznacza liczbę wszystkich wykrytych błędów
X oznacza liczbę posianych błędów, które zostały wykryte
Szacunkowa liczba błędów przed wykonaniem testów: (M - X) * N/X
Szacunkowa liczba błędów po usunięciu wykrytych: (M - X) * (N/X - 1)

Testy systemu

- **Testowanie wstępujące**: najpierw testowane są pojedyncze moduły, następnie moduły coraz wyższego poziomu, aż do osiągnięcia poziomu całego systemu.
- **Testowanie zstępujące**: rozpoczyna się od testowania modułów wyższego poziomu. Moduły niższego poziomu zastępuje się implementacjami szkieletowymi.

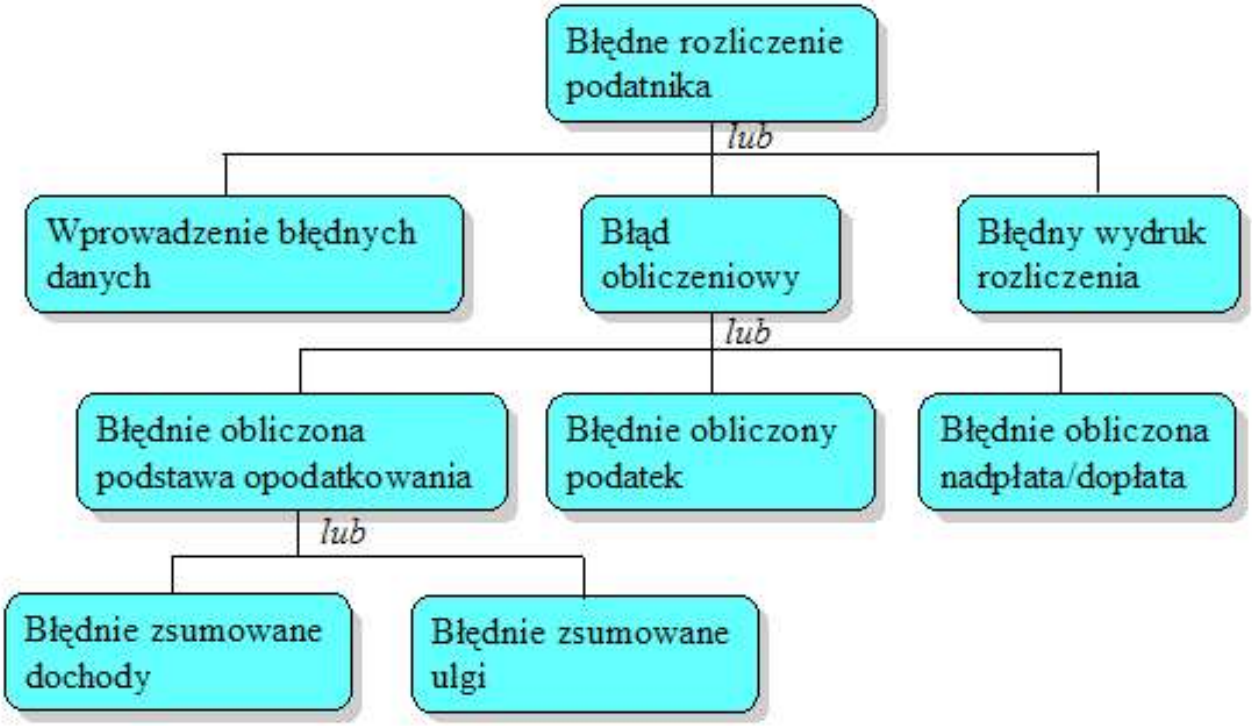
Testy obciążeniowe, odporności

Testy obciążeniowe (*stress testing*). Celem tych testów jest zbadanie wydajności i niezawodności systemu podczas pracy pod pełnym lub nawet nadmiernym obciążeniem. Dotyczy to szczególnie systemów wielodostępnych i sieciowych. Systemy takie muszą spełniać wymagania dotyczące wydajności, liczby użytkowników, liczby transakcji na godzinę. Testy polegają na wymuszeniu obciążenia równego lub większego od maksymalnego.
Testy odporności (*robustness testing*). Celem tych testów jest sprawdzenie działania w przypadku zajścia niepożądanych zdarzeń, np.

- zaniku zasilania
- awarii sprzętowej
- wprowadzenia niepoprawnych danych
- wydania sekwencji niepoprawnych poleceń

Drzewo błędów

Korzeniem drzewa są jest jedna z rozważanych niebezpiecznych sytuacji.
Wierzchołkami są sytuacje pośrednie, które mogą prowadzić do sytuacji odpowiadającej wierzchołkowi wyższego poziomu.



Jakość oprogramowania

Co to jest?

Zapewnienie jakości jest rozumiane jako zespół działań zmierzających do wytworzenia u wszystkich zainteresowanych **przekonania**, że dostarczony produkt właściwie realizuje swoje funkcje i odpowiada aktualnym wymaganiom i standardom. Problem jakości, oprócz mierzalnych czynników technicznych, włącza dużą liczbę niemierzalnych obiektywnie czynników psychologicznych.

Terminologia ISO 9000

jakość - ogół cech i właściwości wyrobu lub usługi decydujący o zdolności wyrobu lub usługi do zaspokojenia stwierdzonych lub przewidywanych potrzeb użytkownika produktu
system jakości - odpowiednio zbudowana struktura organizacyjna z jednoznacznym podziałem odpowiedzialności, określeniem procedur, procesów i zasobów, umożliwiających wdrożenie tzw. *zarządzania jakością*
zarządzanie jakością - jest związane z aspektem całości funkcji zarządzania organizacji, który jest decydujący w określaniu i wdrażaniu *polityki jakości*
polityka jakości - ogół zamierzeń i kierunków działań organizacji dotyczących jakości, w sposób formalny wyrażony przez najwyższe kierownictwo organizacji, będącej systemem jakości
audyt jakości - systematyczne i niezależne badanie, mające określić, czy działania dotyczące jakości i ich wyniki odpowiadają zaplanowanym ustaleniom, czy te ustalenia są skutecznie realizowane i czy pozwalają na osiągnięcie odpowiedniego *poziomu jakości*

Model *jakości*

- **Funkcjonalność**
  - odpowiedniość
  - dokładność
  - współdziałanie
  - zgodność
  - bezpieczeństwo

- **Niezawodność**
  - dojrzałość
  - tolerancja błędów
  - odtwarzalność
- **Użyteczność**
  - zrozumiałość
  - łatwość uczenia
  - łatwość posługiwania się
- **Efektywność**
  - charakterystyka czasowa
  - wykorzystanie zasobów
- **Pielęgnowalność**
  - dostępność
  - podatność na zmiany
  - stabilność
  - łatwość walidacji
- **Przenośność**
  - dostosowywalność
  - instalacyjność
  - zgodność
  - zamienność

Personel ZJO sprawdza czy:

- Projekt jest właściwie zorganizowany, z odpowiednim cyklem życiowym;
- Członkowie zespołu projektowego mają zdefiniowane zadania i odpowiedzialności;
- Plany w zakresie dokumentacji są implementowane;
- Dokumentacja zawiera to, co powinna zawierać;
- Przestrzegane są standardy dokumentacji i kodowania;
- Standardy, praktyki i konwencje są przestrzegane;
- Dane pomiarowe są gromadzone i używane do poprawy produktów i procesów;
- Przeglądy i audyty są przeprowadzane i są właściwie kierowane;
- Testy są specyfikowane i rygorystycznie przeprowadzane;
- Problemy są rejestrowane i reakcja na problemy jest właściwa;
- Projekty używają właściwych narzędzi, technik i metod;
- Oprogramowanie jest przechowywane w kontrolowanych bibliotekach;
- Oprogramowanie jest przechowywane w chroniony i bezpieczny sposób;
- Oprogramowanie od zewnętrznych dostawców spełnia odpowiednie standardy;
- Rejestrowane są wszelkie aktywności związane z oprogramowaniem;
- Personel jest odpowiednio przeszkolony;
- Zagrożenia projektu są zminimalizowane.

Czynniki poprawy jakości

- Poprawa zarządzania projektem
- Wzmocnienie inżynierii wymagań
- Zwiększenie nacisku na jakość
- Zwiększenie nacisku na kwalifikacje i wyszkolenie ludzi
- Szybsze wykonywanie pracy (lepsze narzędzia) 10%
- Bardziej inteligentne wykonywanie pracy (lepsza organizacja i metody) 20%
- Powtórne wykorzystanie pracy już wykonanej (ponowne użycie) 65 %

Plan Zapewnienia Jakości Oprogramowania

Styl. PZJO powinien być zrozumiały, lakoniczny, jasny spójny i modyfikowalny.
Odpowiedzialność. PZJO powinien być wyprodukowany przez komórkę jakości zespołu

podejmujący się produkcji oprogramowania. PZJO powinien być przejrzany i zrecenzowany przez ciało, któremu podlega dana komórka jakości oprogramowania.
Medium. Zwykle PZJO jest dokumentem papierowym. Może być także rozpowszechniony w formie elektronicznej.
Zawartość. PZJO powinien być podzielony na 4 rozdziały, każdy dla następujących faz rozwoju oprogramowania:

- PZJO dla fazy wymagań użytkownika i analizy;
- PZJO dla fazy projektu architektury;
- PZJO dla fazy projektowania i konstrukcji;
- PZJO dla fazy budowy, testowania i instalacji oprogramowania.

Ewolucja. PZJO powinien być tworzony dla następnej fazy po zakończeniu fazy poprzedniej.