1. **Co jest przedmiotem inżynierii oprogramowania? Jakimi zagadnieniami się ona zajmuje?**

Inżynieria oprogramowania jest wiedzą techniczną dotycząca wszystkich faz cyklu życia oprogramowania. Traktuje oprogramowanie jako produkt, który ma spełniać potrzeby techniczne, ekonomiczne lub społeczne.

**Zagadnienia inżynierii oprogramowania**

- Sposoby prowadzenia przedsięwzięć informatycznych.

- Techniki planowania, szacowania kosztów, harmonogramowania i monitorowania przedsięwzięć informatycznych.

- Metody analizy i projektowania systemów.

- Techniki zwiększania niezawodności oprogramowania.

- Sposoby testowania systemów i szacowania niezawodności.

- Sposoby przygotowania dokumentacji technicznej i użytkowej.

- Procedury kontroli jakości.

- Metody redukcji kosztów konserwacji (usuwania błędów, modyfikacji i rozszerzeń)

- Techniki pracy zespołowej i czynniki psychologiczne wpływające na efektywność pracy.

1. **Krótko omów przyczyny kryzysu oprogramowania. Jak można z nimi walczyć?**

- Sprzeczność pomiędzy odpowiedzialnością, jaka spoczywa na współczesnych SI, a ich zawodnością wynikającą ze złożoności i ciągle niedojrzałych metod tworzenia i weryfikacji oprogramowania.

- Ogromne koszty utrzymania oprogramowania.

- Niska kultura ponownego użycia wytworzonych komponentów projektów i oprogramowania; niski stopień powtarzalności poszczególnych przedsięwzięć.

- Długi i kosztowny cykl tworzenia oprogramowania, wysokie prawdopodobieństwo niepowodzenia projektu programistycznego.

- Długi i kosztowny cykl życia SI, wymagający stałych (często globalnych) zmian.

- Eklektyczne, niesystematyczne narzędzia i języki programowania.

Walka z kryzysem:

-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.

1. **Omów pojęcie *modelowanie pojęciowe.***

Projektant i programista muszą dokładnie wyobrazić sobie problem oraz metodę jego rozwiązania. Zasadnicze procesy tworzenia oprogramowania zachodzą w ludzkim umyśle i nie są związane z jakimkolwiek językiem programowania. Modelowanie pojęciowe jest wspomagane przez środki wzmacniające ludzką pamięć i wyobraźnię. Służą one do przedstawienia rzeczywistości opisywanej przez dane, procesów zachodzących w rzeczywistości, struktur danych oraz programów składających się na konstrukcję systemu.

1. **Opisz model spiralny cyklu życia oprogramowania. Oceń jego wady i zalety.**



1. **Opisz model kaskadowy cyklu życia oprogramowania. Oceń jego wady i zalety.**

****

Wady:

-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.

1. **Scharakteryzuj i porównaj model kaskadowy i realizację sterowaną dokumentami. Oceń wady i zalety drugiego z modeli w stosunku do modelu kaskadowego.**

Kaskadowy pyt 5.

Realizacja sterowana dokumentami:

- Przyjęty przez armią amerykańską dla realizacji projektów w języku Ada.

- Jest to odmiana modelu kaskadowego.

- Każda faza kończy się sporządzeniem szeregu dokumentów, w których opisuje się wyniki danej fazy.

- Łatwe planowanie, harmonogramowanie oraz monitorowanie przedsięwzięcia.

 Dodatkowa zaleta: (teoretyczna) możliwość realizacji dalszych faz przez inną firmę

Wady:

- Duży nakład pracy na opracowanie dokumentów zgodnych ze standardem (DOD STD 2167) - ponad 50% całkowitych nakładów.

- Przerwy w realizacji niezbędne dla weryfikacji dokumentów przez klienta.

1. **Scharakteryzuj i porównaj model kaskadowy i prototypowanie. Oceń wady i zalety drugiego z tych modeli w stosunku do modelu kaskadowego.**

Kaskadowy pyt 5.

**Sposób na uniknięcie zbyt wysokich kosztów błędów popełnionych w fazie określania wymagań.** Zalecany w przypadku, gdy określenie początkowych wymagań jest stosunkowo łatwe.

Metody prototypowania:

**Niepełna realizacja**: objęcie tylko części funkcji

**Języki wysokiego poziomu**: Smalltalk, Lisp, Prolog, 4GL, ...

**Wykorzystanie gotowych komponentów**

**Generatory interfejsu użytkownika**: wykonywany jest wyłącznie interfejs, wnętrze systemu jest “podróbką”.

**Szybkie programowanie** (*quick-and-dirty*): normalne programowanie, ale bez zwracania uwagi na niektóre jego elementy, np. zaniechanie testowania

1. **Scharakteryzuj i porównaj model kaskadowy i programowanie odkrywcze. Oceń wady i zalety drugiego z tych modeli w stosunku do modelu kaskadowego.**

Kaskadowy pyt .5

1. **Scharakteryzuj i porównaj model kaskadowy i realizację przyrostową. Oceń wady i zalety drugiego z tych modeli w stosunku do modelu kaskadowego.**

Kaskadowy pyt .5

****

1. **Scharakteryzuj i porównaj model kaskadowy i model spiralny. Oceń wady i zalety drugiego z tych modeli w stosunku do modelu kaskadowego.**

Kaskadowy pyt .5

Spiralny pyt. 4

1. **Jakie kryteria wg Ciebie powinny być brane pod uwagę i jakie czynności powinny zostać dokonane przez podjęciem decyzji o realizacji przedsięwzięcia? W jakiej postaci przedstawione są wyniki tego etapu?**

Odpowiedź na pytanie 13. + poniższe

Udostępniamy klientowi raport, który obejmuje:

* definicję celów przedsięwzięcia
* opis zakresu przedsięwzięcia
* opis systemów zewnętrznych, z którymi system będzie współpracować
* ogólny opis wymagań
* ogólny model systemu
* opis proponowanego rozwiązania
* oszacowanie kosztów
* wstępny harmonogram prac

Raport oceny rozwiązań, zawierający informację o rozważanych rozwiązaniach oraz przyczynach wyboru jednego z nich.

Opis wymaganych zasobów - pracownicy, oprogramowanie, sprzęt, lokale, ...

Definicje standardów.

Harmonogram fazy analizy

1. **Czemu służy faza strategiczna w procesie konstrukcji systemu informacyjnego?**

Faza strategiczna jest wykonywana zanim podejmowana jest decyzja o realizacji przedsięwzięcia.

1. **Jakie czynności są przeprowadzane w trakcie trwania fazy strategicznej?**

Dokonanie serii rozmów (wywiadów) z przedstawicielami klienta

Określenie celów przedsięwzięcia z punktu widzenia klienta

Określenie zakresu oraz kontekstu przedsięwzięcia

Ogólne określenie wymagań, wykonanie zgrubnej analizy i projektu systemu

Propozycja kilku możliwych rozwiązań (sposobów realizacji systemu)

Oszacowanie kosztów oprogramowania

Analiza rozwiązań

Prezentacja wyników fazy strategicznej przedstawicielom klienta oraz korekta wyników

Określenie wstępnego harmonogramu przedsięwzięcia oraz struktury zespołu realizatorów

Określenie standardów, zgodnie z którymi realizowane będzie przedsięwzięcie

1. **Jakie decyzje powinny zostać podjęte w czasie trwania fazy strategicznej?**

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

1. **Wyjaśnij (swoimi słowami), co to jest zakres i kontekst przedsięwzięcia? Podaj własny przykład.**

Zakres przedsięwzięcia: 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.

Kontekst przedsięwzięcia: systemy, organizacje, użytkownicy zewnętrzni, z którymi tworzony system ma współpracować.

Program podatkowy:

**Zakresem** przedsięwzięcia jest działalność jednej firmy rachunkowej, która może mieć dowolną liczbę klientów. Nie jest określone, czy system ma drukować wypełniony PIT, czy tylko dostarczać dane.

Pracownik firmy jest jedynym **systemem zewnętrznym**.

1. **Jakie elementy wpływają na ocenę rozwiązania wybranego w fazie strategicznej, w jakich jednostkach można je określić i w jaki sposób można dokonać wyboru między kilkoma rozwiązaniami?**

koszt

 czas realizacji

 niezawodność

 możliwość ponownego użycia

 przenośność na inne platformy

 wydajność (szybkość)

Sposoby:

**Usunięcie** rozwiązań zdominowanych, tj. gorszych wg wszystkich kryteriów

(lub prawie wszystkich).

**Normalizacja** wartości dla poszczególnych kryteriów (sprowadzenie do przedziału [0,1])

**Przypisanie wag** do kryteriów (również może być trudne).

1. **Co wpływa na koszty konstruowanego oprogramowania i w jaki sposób dokonuje się oszacowania poszczególnych składników?**

koszt sprzętu będącego częścią tworzonego systemu

 koszt wyjazdów i szkoleń

 koszt zakupu narzędzi

 nakład pracy

Techniki szacowania:

**Modele algorytmiczne**. Wymagają opisu przedsięwzięcia przez wiele atrybutów liczbowych i/lub opisowych. Odpowiedni algorytm lub formuła matematyczna daje wynik.

**Ocena przez eksperta**. Doświadczone osoby z dużą precyzją potrafią oszacować koszt realizacji nowego systemu.

**Ocena przez analogię** (historyczna). Wymaga dostępu do informacji o poprzednio realizowanych przedsięwzięciach. Metoda podlega na wyszukaniu przedsięwzięcia o najbardziej zbliżonych charakterystykach do aktualnie rozważanego i znanym koszcie i następnie, oszacowanie ewentualnych różnic.

**Wycena dla wygranej**. Koszt oprogramowania jest oszacowany na podstawie kosztu oczekiwanego przez klienta i na podstawie kosztów podawanych przez konkurencję.

**Szacowanie wstępujące**. Przedsięwzięcie dzieli się na mniejsze zadania, następnie sumuje się koszt poszczególnych zadań.

1. **Omów do czego służą metody FPA i COCOMO, jaki jest między nimi związek?**

COCOMO jest oparte na kilku formułach pozwalających oszacować całkowity koszt przedsięwzięcia na podstawie oszacowanej liczby linii kodu.

**Metoda punktów funkcyjnych oszacowuje koszt projektu na podstawie funkcji użytkowych, które system ma realizować.** Stąd wynika, ze metoda ta może być stosowana dopiero wtedy, gdy funkcje te są z grubsza znane.

Istnieją przeliczniki punktów funkcyjnych na liczbę linii kodu, co może być podstawą dla metody COCOMO.

1. **W jakim celu wykorzystuje się drzewa ryzyka? Podaj własny przykład takiego drzewa i krótko go omów.**

W celu wyboru rozwiązania.



1. **Jakie znasz algorytmiczne modele kosztów i jaki rodzaj kosztów one liczą?**

**COCOMO** jest oparte na kilku formułach pozwalających oszacować całkowity koszt przedsięwzięcia na podstawie oszacowanej liczby linii kodu.

**Metoda punktów funkcyjnych** oszacowuje koszt projektu na podstawie funkcji użytkowych, które system ma realizować. Stąd wynika, ze metoda ta może być stosowana dopiero wtedy, gdy funkcje te są z grubsza znane.

1. **Omów, co jest celem fazy określenia wymagań.**

Celem fazy określenia wymagań jest ustalenie wymagań klienta wobec tworzonego systemu. Dokonywana jest zamiana celów klienta na konkretne wymagania zapewniające osiągnięcie tych celów.

1. **Z czego wynikają trudności związane z określeniem wymagań użytkownika?**

Klient z reguły nie wie dokładnie w jaki sposób osiągnąć założone cele.

Cele klienta mogą być osiągnięte na wiele sposobów.

Duże systemy są wykorzystywane przez wielu użytkowników. Ich cele są często sprzeczne. Różni użytkownicy mogą posługiwać się inną terminologią mówiąc o tych samych problemach.

Zleceniodawcy i użytkownicy to często inne osoby. Głos zleceniodawców może być w tej fazie decydujący, chociaż nie zawsze potrafią oni właściwie przewidzieć potrzeby przyszłych użytkowników.

1. **Kto i w oparciu o co rozpoznaje wymagania na konstruowany system?**

Faza określenia wymagań wymaga dużego zaangażowania ze strony klienta, ze strony przyszłych użytkowników systemu i ekspertów w dziedzinie.

Ta faza nie jest więc prostym zbieraniem wymagań, lecz procesem, w którym klient wspólnie z przedstawicielem producenta konstruuje zbiór wymagań zgodnie z postawionymi celami.

1. **Jakie znasz metody rozpoznania wymagań na konstruowany system? Krótko je omów.**

**Wywiady i przeglądy.** Wywiady powinny być przygotowane (w postaci listy pytań) i podzielone na odrębne zagadnienia. Podział powinien przykrywać całość tematu i powinny być przeprowadzone na reprezentatywnej grupie użytkowników. Wywiady powinny doprowadzić do szerokiej zgody i akceptacji projektu.

**Studia** **na istniejącym oprogramowaniem**. Dość często nowe oprogramowanie zastępuje stare. Studia powinny ustalić wszystkie dobre i złe strony starego oprogramowania.

**Studia wymagań systemowych**. Dotyczy sytuacji, kiedy nowy system ma być częścią większego systemu.

**Studia osiągalności.** Określenie realistycznych celów systemu i metod ich osiągnięcia.

**Prototypowanie**. Zbudowanie prototypu systemu działającego w zmniejszonej skali, z uproszczonymi interfejsami.

1. **Jakie znasz metody specyfikacji wymagań na konstruowany system? Krótko je omów.**

**Język naturalny** - najczęściej stosowany. Wady: niejednoznaczność powodująca różne rozumienie tego samego tekstu; elastyczność, powodująca wyrazić te same treści na wiele sposobów. Utrudnia to wykrycie powiązanych wymagań i powoduje trudności w wykryciu sprzeczności.

**Formalizm matematyczny**. Stosuje się rzadko (dla specyficznych celów).

**Język naturalny strukturalny**. Język naturalny z ograniczonym słownictwem i składnią. Tematy i zagadnienia wyspecyfikowane w punktach i podpunktach.

**Tablice, formularze**. Wyspecyfikowanie wymagań w postaci (zwykle dwuwymiarowych) tablic, kojarzących różne aspekty (np. tablica ustalająca zależność pomiędzy typem użytkownika i rodzajem usługi).

**Diagramy blokowe:** forma graficzna pokazująca cykl przetwarzania.

**Diagramy kontekstowe:** ukazują system w postaci jednego bloku oraz jego powiązania z otoczeniem, wejściem i wyjściem.

**Diagramy przypadków użycia**: poglądowy sposób przedstawienia aktorów i funkcji systemu.

1. **Wymień w punktach i krótko omów czynniki, które należy uwzględnić przy konstruowaniu wymagań.**

**Możliwości systemu**: Zestaw funkcji, które ma wykonywać system, uporządkowany hierarchicznie.

**Objętość**: Ilu użytkowników będzie pracować jednocześnie? Ile terminali ma być podłączone do systemu? Ile czujników będzie kontrolowanych jednocześnie? Ile danych będzie przechowywane?

**Szybkość**: Jak długo może trwać operacja lub sekwencja operacji? Liczba operacji na jednostkę czasu. Średni czas niezbędny dla jednej operacji.

**Dokładność**: Określenie stopnia precyzji pomiarów lub przetwarzania. Określenie wymaganej dokładności wyników. Zastąpienie wyników ilościowych jakościowymi lub odwrotnie.

**Ograniczenia**: ograniczenia na interfejsy, jakość, skalę czasową, sprzęt, oprogramowanie, skalowalność, itd.

**Interfejsy komunikacyjne**: sieć, protokoły, wydajność sieci, poziom abstrakcji protokołów komunikacyjnych, itd.

**Interfejsy sprzętowe**: specyfikacja wszystkich elementów sprzętowych, które będą składały się na system, fizyczne ograniczenia (rozmiar, waga), wydajność (szybkość, RAM, dysk, inne pamięci), wymagania co do powierzchni lokalowych, wilgotności, temperatury i ciśnienia, itd.

**Interfejsy oprogramowania**: Określenie zgodności z innym oprogramowaniem, określenie systemów operacyjnych, języków programowania, kompilatorów, edytorów, systemów zarządzania bazą danych, itd.

**Interakcja człowiek-maszyna**: Wszystkie aspekty interfejsu użytkownika, rodzaj języka interakcji, rodzaj sprzętu (monitor, mysz, klawiatura), określenie formatów (układu raportów i ich zawartości), określenie komunikatów dla użytkowników (język, forma), pomocy, komunikatów o błędach, itd.

**Adaptowalność**: Określenie w jaki sposób będzie organizowana reakcja na zmiany wymagań: dodanie nowej komendy, dodanie nowego okna interakcji, itd.

**Bezpieczeństwo**: założenia co do poufności, prywatności, integralności, odporności na hakerów, wirusy, wandalizm, sabotaż, itd.

**Odporność na awarie**: konsekwencje błędów w oprogramowaniu, przerwy w zasilaniu, kopie zabezpieczające, częstotliwości składowania, dziennika zmian, itd.

**Standardy**: Określenie dokumentów standardyzacyjnych, które mają zastosowanie do systemu: formaty plików, normy czcionek, polonizacja, standardy procesów i produktów, itd.

**Zasoby**: Określenie ograniczeń finansowych, ludzkich i materiałowych.

**Skala czasowa**: ograniczenia na czas wykonania systemu, czas szkolenia, wdrażania, itd.

1. **Co powinien zawierać poprawny dokument wymagań użytkownika? Krótko scharakteryzuj poszczególne elementy takiego dokumentu.**

Wymagania powinny być zebrane w dokumencie - opisie wymagań.

Dokument ten powinien być podstawą szczegółowego kontraktu między klientem a producentem oprogramowania.

Powinien także pozwalać na weryfikację stwierdzającą, czy wykonany system rzeczywiście spełnia postawione wymagania.

Powinien to być dokument zrozumiały dla obydwu stron.

1. **Jaki poziom ogólności i jakie cechy powinny charakteryzować dobry opis wymagań na konstruowany system?**

Jasność: jednoznaczne sformułowania, zrozumiały dla użytkowników i projektantów. Strukturalna organizacja dokumentu.

Spójność: brak konfliktów w wymaganiach.

Modyfikowalność: wszystkie wymagania są sformułowane w jasnych punktach, które mogą być wyizolowane z kontekstu i zastąpione przez inne.

Możliwość dodawania nowych wymagań, możliwość ich modyfikacji

1. **Krótko scharakteryzuj i porównaj wymagania funkcjonalne i niefunkcjonalne.**

Wymagania funckjonalne: Opisują funkcje (czynności, operacje) wykonywane przez system.

Funkcje te mogą być również wykonywane przy użyciu systemów zewnętrznych.

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.

1. **Wyjaśnij, w jaki sposób dokumentuje się wymagania funkcjonalne i niefunkcjonalne? Podaj własne przykłady.**
2. **Co jest celem fazy analizy, a co jest jej wynikiem?**

Celem fazy analizy jest ustalenie wszystkich tych czynników lub warunków w dziedzinie przedmiotowej, w otoczeniu realizatorów projektu, w istniejących lub planowanych systemach komputerowych, które mogą wpłynąć na decyzje projektowe, na przebieg procesu projektowego i na realizację wymagań.

Wynikiem jest logiczny model systemu, opisujący sposób realizacji przez system postawionych wymagań, lecz abstrahujących od szczegółów implementacyjnych.

1. **Wyjaśnij pojęcie model analityczny. Jakimi cechami charakteryzuje się ten model?**

Logiczny model oprogramowania:

* + pokazuje co system musi robić;
	+ jest zorganizowany hierarchicznie, wg poziomów abstrakcji
	+ unika terminologii implementacyjnej
	+ pozwala na wnioskowanie „od przyczyny do skutku” i odwrotnie.

Cechy:

Uproszczony opis systemu;

Hierarchiczna dekompozycja funkcji systemu;

Model logiczny jest opisany przy pomocy notacji zgodnej z pewną konwencją;

Jest on zbudowany przy użyciu dobrze rozpoznanych metod i narzędzi;

Jest on używany do wnioskowania o przyszłym oprogramowaniu;

1. **Jakie czynności są przeprowadzone w trakcie trwania fazy analizy?**

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.

1. **Wyjaśnij pojęcie systemu zewnętrznego. Czy model będący wynikiem analizy powinien je ujmować?**
2. **Jakie znasz rodzaje notacji. Która z nich najlepiej nadaje się do komunikacji między ludźmi?**

Język naturalny

 Notacje graficzne

 Specyfikacje - ustrukturalizowany zapis tekstowy i numeryczny

1. **Wyjaśnij różnicę między notacją a metodyką, podaj przykłady.**

**Różnice: składnia, semantyka, pragmatyka**

**Metodyka** jest zbiorem zasad dotyczących sposobów wykonywania jakiejś pracy lub trybu postępowania prowadzącego do określonego celu wraz ze zbiorem narzędzi przeznaczonych do wykonania tej pracy oraz wiedzą w jaki sposób posługiwać się tymi zasadami i narzędziami.

**Notacja** opisu metodyki oparta jest na dwóch podstawowych modelach języka UML: modelu klas oraz modelu czynności. To umowny sposób zapisu symboli, liter, znaków, itp. Notacja umożliwia w sposób formalny zapis treści wyrażeń reguł, wzorów, formuł, itd.

1. **Czym się różnią metodyki strukturalne od obiektowych?**

Metodyki strukturalne - łączą statyczny opis danych oraz statyczny opis procesów.

analiza strukturalna używa następujących technik.

* + Diagramy Przepływu Danych (*Data Flow Diagrams, DFD*)
	+ Słownik Danych (*Data Dictionary*)
	+ Strukturalny Angielski (*Structured English*) -> strukturalny polski
	+ Tablice Decyzyjne (*Decision Tables*)
	+ Drzewa Decyzyjne (*Decision Trees*)

Metodyka wykorzystująca pojęcia obiektowości dla celów modelowania pojęciowego oraz analizy i projektowania systemów informatycznych.

Podstawowym składnikiem jest diagram klas, będący zwykle wariantem notacyjnym i pewnym rozszerzeniem diagramów encja-związek.

Diagram klas zawiera: klasy, w ramach klas specyfikacje atrybutów i metod, związki generalizacji, związki asocjacji i agregacji, liczności tych związków, różnorodne ograniczenia oraz inne oznaczenia.

Uzupełnieniem tego diagramu są inne: diagramy dynamiczne uwzględniające stany i przejścia pomiędzy tymi stanami, diagramy interakcji ustalające zależności pomiędzy wywołaniami metod, diagramy funkcjonalne (będące zwykle pewną mutacją diagramów przepływu danych), itd.

Koncepcja przypadków użycia (*use cases*) zakłada odwzorowanie struktury systemu z punktu widzenia jego użytkownika.

**4.8 Omów pojęcie “metodyki obiektowe”. Wymień znane Ci metodyki obiektowe.**

**Metodyki obiektowe-** jest to metodyka wykorzystująca pojęcia obiektowości dla celów modelowania pojęciowego oraz analizy i projektowania systemów informatycznych. Podstawowym składnikiem jest diagram klas będący zwykle wariantem notacyjnym i pewnym rozszerzeniem diagramów encja-związek.Diagram klas zawiera:klasy,atrybuty,metody,zwiazki generalizacji,związki asocjacji,agregacji,liczności tych związków,różnorodne ograniczenia oraz inne oznaczenia. Uzupełnieniem tego diagramu są inne: diagramy dynamiczne (uwzględniające stany i przejścia pomiędzy stanami),diagramy interakcji (ustalające zależności pomiędzy wywołaniami metod), diagramy funkcjonalne (będące zwykle pewną mutacją diagramów przepływu danych) itd. Koncepcja przypadku użycia zakłada odwzorowanie struktury systemu z punktu widzenia użytkownika.

**Przykłady metodyk obiektowych:** Express, OODA(Booch), OMT(Rumbaugh), OOSA(Shlaer-Mellor), Objectory(Jacobson), MOSES/OPEN, OOA/OOD(Coad/Yourdon), Notacja UML, RUP.

**4.9 Jakie zadania muszą być zrealizowane w trakcie budowania modelu obiektowego ?**

 **Zadania do zrealizowania w trakcie budowania modelu obiektowego:**

-Identyfikacja klas i obiektów

-Identyfikacja związków pomiędzy klasami

-Identyfikacja i definiowanie pól (atrybutów)

-Identyfikacja i definiowanie metod i komunikatów

Czynności te są wykonywane iteracyjnie. Kolejność ich wykonywania nie jest ustalona i zależy zarówno od stylu pracy, jak i od konkretnego problemu.

**4.10 Omów podstawowe rezultaty fazy analizy.**

**Podstawowe rezultaty fazy analizy:**

-Poprawiony dokument opisujący wymagania

-Słownik danych zawierający specyfikację modelu

-Dokument opisujący stworzony model, zawierający:

1. Diagram klas
2. Diagram przypadków użycia
3. Diagramy sekwencji komunikatów (dla wybranych sytuacji)
4. Raport zawierający definicje i opisy klas, atrybutów, związków, metod, itd.

-Harmonogram fazy projektowania

-Wstępne przypisanie ludzi i zespołów do zadań

**5.1** **Omów co jest celem fazy projektowania ? Jak się ma okres jej trwania do okresów trwania faz : strategicznej,analizy,określania wymagań?**

**Celem fazy projektowania** jest opracowanie szczegółowego opisu implementacji systemu. **Okres Trwania fazy projektowania** jest taki sam jak w fazie strategicznej ,dłuższy niż faza analizy i krótszy niż faza określenia wymagań.Faza projektowania jest po fazie strategicznej i po określeniu wymagań natomiast faza Analizy zawiera w sobie część fazy Projektowania.

**5.2 Omów zadania, jakie powinny być wykonane podczas trwania fazy projektowania.**

**Zadania w fazie Projektowania:**

-Uszczegółowienie wyników analizy. Projekt musi być wystarczająco szczegółowy aby mógł być podstawą implementacji. Stopień szczegółowości zależy od poziomu zaawansowania programistów.

-Projektowanie składowych systemów nie związanych z dziedziną problemu

-Optymalizacja systemu

-Dostosowanie do ograniczeń i możliwości środowiska implementacji

-Określenie fizycznej struktury systemu.

**5.3 Czym różnią się wyniki fazy projektowania od wyników fazy analizy ?**

**Różnice wyników fazy projektowania a fazy analizy:**

W odróżnieniu od analizy, w projektowaniu dużą role odgrywa środowisko implementacji. Projektanci muszą więc posiadać dobrą znajomość języków, bibliotek, i narzędzi stosowanych w trakcie implementacji.

**5.4 Który z modeli (faza analizy czy faza projektowania ) uwzględnia elementy systemu nie związane z dziedziną problemu ? Jakie to elementy ?**

**Faza Projektowania uwzględnia składowe systemów nie związane z dziedziną problemu** i są nią elementy:

-składowej interfejsu użytkownika

-składowej zarządzania danymi (przechowywanie trwałych danych)

-składowej zarządzania pamięcią operacyjną

- składowej zarządzania zadaniami (podział czasu procesora)

**5.5 W jaki sposób w fazie projektowania precyzuje się wyniki analizy ? Krótko omów te sposoby.**

**Sposoby precyzowania wyników analizy w fazie Projektowania:**

-Uszczegółowianie poprzez podanie reguł odwzorowanie notacji

w struktury języka programowania.

-Uszczegółowianie metod:

1. Podanie nagłówków metod oraz ich parametrów.
2. Określenie, które z metod będą realizowane jako funkcje wirtualne (późno wiązane, podczas wykonania) a które jako zwyczajne funkcje (wiązane statycznie podczas kompilacji i linkowania).
3. Zastąpienie niektórych prostych metod bezpośrednim dostępem do atrybutów.

Np. metody PobierzNazwisko, UstawNazwisko, etc.

1. Zastąpienie niektórych atrybutów redundantnych przez odpowiednie metody, np.

Wiek = BieżącaData - DataUrodzenia;

KwotaDochodu = KwotaPrzychodu - KwotaKosztów;

-Określenie sposobów implementacji związków (asocjacji)

Związki można zaimplementować na wiele sposobów, z reguły poprzez wprowadzenie dodatkowych atrybutów (pól). Mogą one być następujące:

1. obiekty powiązanej klasy
2. wskaźniki (referencje) do obiektów powiązanej klasy
3. identyfikatory obiektów powiązanej klasy
4. klucze kandydujące obiektów powiązanej klasy

W zależności od przyjętego sposobu oraz od liczności związków (1:1, 1:n, n:1, m:n) możliwe są bardzo różne deklaracje w przyjętym języku programowania.

**5.6 Wyjaśnij, czego dotyczy pojęcie RAD? Podaj przykłady takich narzędzi.**

**RAD**-Rapid Application Development (Szybkie rozwijanie aplikacji )

Terminem tym określa się narzędzia i techniki programowania umożliwiające szybką budowę prototypów lub gotowych aplikacji, z reguły oparte o programowanie wizyjne. Termin RAD występuje niekiedy jako synonim języków/środowisk czwartej generacji (4GL).

Przykładami narzędzi RAD są: Borland Delphi RAD Pack, IBM VisualAge (for Cobol, Java, C++, Smalltalk), Microsoft Access Developer’s Toolkit, Microsoft Visual FoxPro Professional, PowerBuilder Desktop, Power++ i wiele innych

**5.7 Które ze składowych systemu w najmniejszym, a które w największym stopniu poddają się automatyzacji w fazie projektowania ? Wymień znane Ci narzędzia wspomagające automatyzację.**

**Najmniej poddaje się automatyzacji** Składowa dziedziny problemu.Niekiedy inne ograniczenia lub nietypowość wykluczają możliwość zastosowania narzędzi RAD.

Najbardziej poddaje się automatyzacji

**5.8 Jakie są podstawowe reguły, w oparciu o które tworzy się w fazie projektowania interfejs użytkownika ?**

**Podstawowe reguły w oparciu o które tworzy się w fazie projektowania interfejs użytkownika:**

**-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:

1. Umieszczanie etykiet zawsze nad lub obok pól edycyjnych
2. Umieszczanie typowych pól OK i Anuluj zawsze od dołu lub od prawej.
3. 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:** Reguła ta określa, że człowiek może się jednocześnie wydajnie 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.

**5.9 W jaki sposób organizowałbyś interakcje systemu z użytkownikiem ? Na jakie elementy należy zwrócić uwage ?**

**Organizacja interakcji systemu z użytkownikiem:**

**-Za pomocą linii komend.** Elementy na które trzeba zwrócic uwage:

1. Są dla niewielkich systemów.
2. Są dla prototypów.
3. Są dla zaawansowanych użytkowników.
4. Często szybszy od niż interfejs pełnoekranowy.

-**W pełnoekranowym środowisku okienkowym.** Elementy na które trzeba zwrócic uwage:

1. Dla dużych systemów.
2. Wygodny dla początkujących i średnio zaawansowanych użytkowników

**5.10 Omów, jakie składowe systemu (nie związane z dziedziną problemu) należy uwzględnić w fazie projektowania ?**

**Składowe systemu nie związane z dziedzina problemu które trzeba uwzględnić:**

-składowa interfejsu użytkownika

-składowa zarządzania danymi (przechowywanie trwałych danych)

-składowa zarządzania pamięcią operacyjną

-składowa zarządzania zadaniami (podział czasu procesora)

**6.1 Omów podstawowe rezultaty fazy projektowania.**

 **Podstawowe rezultaty fazy projektowania**

-Poprawiony dokument opisujący wymagania

-Poprawiony model

-Uszczegółowiona specyfikacja projektu zawarta w słowniku danych

-Dokument opisujący stworzony projekt składający się z (dla obiektowych):

1. diagramu klas
2. diagramów interakcji obiektów
3. diagramów stanów
4. innych diagramów, np. diagramów modułów, konfiguracji
5. zestawień zawierających:
6. definicje klas
7. definicje atrybutów
8. definicje danych złożonych i elementarnych
9. definicje metod

-Zasoby interfejsu użytkownika, np. menu, dialogi

-Projekt bazy danych

-Projekt fizycznej struktury systemu

-Poprawiony plan testów

-Harmonogram fazy implementacji

**6.2 Omów wymagania niefunkcjonalne ,określane dla fazy projektowania.**

**Wymagania niefunkcjonalne dla fazy projektowania**

- Wymagania odnośnie wydajności

- Wymagania odnośnie interfejsu (protokoły, formaty plików, ...)

- Wymagania operacyjne (aspekty ergonomiczne, języki, pomoce)

- Wymagania zasobów (ilość procesorów, pojemność dysków, ...)

- Wymagania w zakresie weryfikacji (sposoby przeprowadzenia)

- Wymagania w zakresie akceptacji i testowania

- Wymagania odnośnie dokumentacji

- Wymagania odnośnie bezpieczeństwa

- Wymagania odnośnie przenaszalności

- Wymagania odnośnie jakości

1. wybór metod projektowania
2. decyzje dotyczące ponownego użycia
3. wybór narzędzi
4. wybór metod oceny projektu przez ciała zewnętrzne

- Wymagania odnośnie niezawodności

- Wymagania odnośnie podatności na pielęgnację (maintenance)

- Wymagania odnośnie odporności na awarie

**6.3 Omów wykorzystanie narzędzi CASE w fazie projektowania.**

**Narzędzia CASE w fazie projektowania**

Tradycyjnie stosuje się Lower-CASE (projektowanie struktur logicznych).

1. Edytor notacji graficznych
2. Narzędzia edycji słownika danych
3. Generatory raportów
4. Generatory dokumentacji technicznej
5. Narzędzia sprawdzania jakości projektu

- Narzędzia CASE powinny wspomagać proces uszczegóławiania wyników analizy. Powinny np. automatycznie dodawać atrybuty realizujące związki pomiędzy klasami. Powinny ułatwiać dostosowanie projektu do środowiska implementacji.

- Powinna istnieć możliwość automatycznej transformacji z modelu obiektów na schemat relacyjnej bazy danych.

- Niektóre narzędzia CASE umożliwiają projektowanie interfejsu użytkownika.

- Narzędzia inżynierii odwrotnej (reverse engineering), dla odtworzenia projektu na podstawie istniejącego kodu.

**6.4 Jakie reguły należy uwzględnić przy projektowaniu składowej zarządzania danymi ?**

**Reguły projektowania składowej zarządzania danymi:**

-Trwałe dane mogą być przechowane w:

1. pliku
2. w bazie danych (relacyjnej, obiektowej, lub innej).

-Poszczególne elementy danych - zestawy obiektów lub krotek - mogą być przechowywane w następującej postaci:

1. w jednej relacji lub pliku
2. w odrębnym pliku dla każdego rodzaju obiektów lub krotek

-Sprowadzenie danych do pamięci operacyjnej oraz zapisanie do trwałej pamięci może być:

1. na bieżąco, kiedy program zażąda dostępu i kiedy następuje zapełnienie bufora
2. na zlecenie użytkownika

**6.5 Jakie problemy pojawiają się w momencie przechodzenia z pojęciowego modelu obiektowego na relacyjny model logiczny ?**

**Problemy przy przechodzeniu z pojęciowego modelu obiektowego na relacyjny model logiczny:**

* Konieczność przeprowadzenie nietrywialnych odwzorowań przy przejściu z modelu pojęciowego (np. w UML) na strukturę relacyjną.
* Ustalony format krotki (rekordu) może powodować trudności przy polach zmiennej długości (w niektórych systemach).
* Trudności (niesystematyczność) reprezentacji dużych wartości (grafiki, plików tekstowych, itd.). Zwykle niestandardowa.
* W niektórych sytuacjach - duże narzuty na czas przetwarzania (kosztowne złączenia)
* 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)

**6.6 Na czym polega i czemu służy optymalizacja projektu ?**

**Optymalizacja projektu** polega na zwiększeniu wydajności zaimplementowanego kodu.

Bezpośrednia implementacja projektu może prowadzić do systemu o zbyt niskiej efektywności.

Optymalizacja może być dokonana:

1. Na poziomie projektu
2. Na poziomie implementacji

**Sposoby optymalizacji:**

-Sprawdzenie efektywności algorytmów przetwarzania. Np. zmiana algorytmu sortującego poprzez wprowadzenie pośredniego pliku zawierającego tylko klucze i wskaźniki do sortowanych obiektów może przynieść nawet 100-krotny zysk.

-Wyłowienie “wąskich gardeł” w przetwarzaniu i optymalizacja tych wąskich gardeł poprzez starannie rozpracowane procedury. Znana jest teza, że 20% kodu jest wykonywane przez 80% czasu (zasada Pareta).

-Zaprogramowanie “wąskich gardeł” w języku niższego poziomu, np. w C dla programów w 4GL.

-Denormalizacja relacyjnej bazy danych, łączenie dwóch lub więcej tablic w jedną. Wbrew „teorii”, która nakazuje normalizację.

-Stosowanie indeksów, tablic wskaźników i innych struktur pomocniczych.

-Analiza mechanizmów buforowania danych w pamięci operacyjnej i

ewentualna zmiana tego mechanizmu (np. zmniejszenie liczby poziomów)

**6.7 Jakie rodzaje ograniczeń mogą wystąpić w środowisku implementacji i które z nich można obejść ?**

**Rodzaje ograniczenia w srodowisku implementacji:**

- Brak dziedziczenia wielokrotnego.

Można obejść za pomocą powtórzenia atrybutów i metod obu nadklas.

- Brak dziedziczenia.

 Można obejść

- Brak metod wirtualnych (przesłaniania).

Można obejść

- Brak złożonych atrybutów

 Można obejść

- Brak typów multimedialnych

**6.8 Omów ,na czym polega poprawność projektu.**

**Poprawność projektu -** oznacza, że opis projektu jest zgodny z zasadami posługiwania się notacjami. Nie gwarantuje, że projekt jest zgodny z wymaganiami użytkownika

Poprawny projekt musi być:

1. kompletny
2. niesprzeczny
3. spójny
4. zgodny z regułami składniowymi notacji

**6.9 Omów ,na czym polega spójność projektu.**

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

**6.10 Na czym polega stopień powiązania składowych i jaki powinien być w dobrym projekcie ?**

**Stopień powiązania składowych** -To kryterium określa podział projektu na części zaś oprogramowanie na moduły. W dobrym projekcie powinno dążyć się do tego, aby stopień powiązania pomiędzy jego składowymi był minimalny.

Przykłady powiązań pomiędzy składowymi:

1. Korzystanie przez procesy/moduły z tych samych danych
2. Przepływy danych pomiędzy procesami/modułami
3. Związki pomiędzy klasami
4. Przepływy komunikatów
5. Dziedziczenie

Stopień powiązań można oceniać przy pomocy miar liczbowych (kohezja).

**7.1** **Omów Znaczenie niezawodności oprogramowania**

Znaczenie niezawodności:

-Rosnące oczekiwania klientów wynikające m.in. z wysokiej niezawodności sprzętu. Niemniej nadal niezawodność oprogramowania znacznie ustępuje niezawodności sprzętu. Jest to prawdopodobnie nieuchronne ze względu na znacznie mniejszą powtarzalność oprogramowania i stopień jego złożoności.

-Potencjalnie duże koszty błędnych wykonań, wysokie straty finansowe wynikające z błędnego działania funkcji oprogramowania, nawet zagrożenie dla życia.

-Nieprzewidywalność efektów oraz trudność usunięcia błędów w oprogramowaniu. Często pojawia się konieczność znalezienia kompromisu pomiędzy efektywnością i niezawodnością. Łatwiej jednak pokonać problemy zbyt małej efektywności niż zbyt małej niezawodności.

**7.2 Omów krótko istotę różnicy między unikaniem błędów a tolerancją błędów w oprogramowaniu**

Żadna technika nie gwarantuje uzyskania programu w pełni bezbłędnego.

Tolerancja błędów oznacza, że program działa poprawnie, a przynajmniej sensownie także wtedy, kiedy zawiera błędy.

**7.3 Z jakim pojęciem (unikania błędów czy tolerancji błędów) wiążą się niebezpieczne techniki programowania ? Podaj i krótko omów co najmniej pięć takich technik**

Pojęcie niebezpieczne techniki programowania wiążą się z unikaniem błędów.

Przykłady niebezpiecznych technik:

* Instrukcja go to prowadząca do programów, których działanie jest trudne do zrozumienia.
* Stosowanie licz ze zmiennym przecinkiem, których dokładność jest ograniczona i może być przyczyną nieoczekiwanych błędów
* Wskaźniki i arytmetyka wskaźników: technika wyjątkowo niebezpieczna, dająca możliwość dowolnej penetracji całej pamięci operacyjnej i dowolnych nieoczekiwanych zmian w tej pamięci.
* Obliczenia równoległe. Prowadzą do złożonych zależności czasowych i tzw. pogoni (zależności wyniku od losowego faktu, który z procesów szybciej dojdzie do pewnego punktu w obliczeniach). Bardzo trudne do testowania. Modne wątki są bardzo niebezpieczne i określane są jako zatrute jabłko.
* Przerwania i wyjątki. Technika ta wprowadza pewien rodzaj równoległości, powoduje problemy j/w. Dodatkowo, ryzyko zawieszenia programu.
* Rekurencja. Trudna do zrozumienia, utrudnia śledzenie programu, może losowo powodować przepełnienie stosu wołań (call stack)
* Złożone wyrażenia bez form nawiasowych: korzystanie z priorytetu operatorów, który zwykle jest trudny do skontrolowania przez programistę.

**7.4 W jaki sposób można całkowicie uniknąć błędów w oprogramowaniu, a jakie zalecenia pomagają zmniejszyć wystąpienie błędu ?**

Pełne uniknięcie błędów jest niemożliwe. Można znacznie zmniejszyć prawdopodobieństwo wystąpienia błędu stosując następujące zalecenia:

* Unikanie niebezpiecznych technik (np. programowanie poprzez wskaźniki)
* Stosowanie zasady ograniczonego dostępu (reguły zakresu, hermetyzacja, podział pamięci, itd.)
* Zastosowanie języków z mocną kontrolą typów i kompilatorów sprawdzających zgodność typów
* Stosowanie języków o wyższym poziomie abstrakcji
* Dokładne i konsekwentne specyfikowanie interfejsów pomiędzy modułami oprogramowania
* Szczególna uwaga na sytuacje skrajne (puste zbiory, pętle z zerową ilością obiegów, wartości zerowe, niezainicjowane zmienne, itd.)
* Wykorzystanie gotowych komponentów (np. gotowych bibliotek procedur lub klas) raczej niż pisanie nowych (ponowne użycie, reuse)
* Minimalizacja różnic pomiędzy modelem pojęciowym i modelem implementacyjnym

**7.5 Co rozumiesz pod pojęciem typu i co było zasadniczym celem jego wprowadzenia ?**

**Typ** jest wyrażeniem (oraz pewną abstrakcją programistyczną) przypisaną do pewnych bytów programistycznych, np. zmiennej, danej, obiektu, funkcji, procedury, operacji, metody, parametru procedury, modułu, ADT, wyjątku, zdarzenia.

Typ specyfikuje rodzaj wartości, które może przybierać byt programistyczny, lub „zewnętrzne” cechy tego bytu (interfejs).

Typ jest formalnym ograniczeniem narzuconym na budowę zmiennych lub obiektów. Typy określają również parametry i wyniki procedur, funkcji i metod.

Typ stanowi ograniczenie kontekstu, w którym odwołanie do danego bytu programistycznego może być użyte w programie.

**Zasadniczym celem typów** jest kontrola formalnej poprawności programu.

Ważnym celem typów jest wspomaganie modelowania pojęciowego. Nazwa typu zwykle przenosi nieformalna semantykę zmiennej lub obiektu, któremu jest ten typ przypisany; np. typem zmiennej D jest typ Data.

**7.6 Krótko omów mocną kontrolę typu**

W językach z mocnym typowaniem (strong typing) każdy deklarowany byt programistyczny musi być obowiązkowo wyposażony w deklarację typu. Poprzez deklarację programista wyraża oczekiwania co do roli tego bytu Jest to następnie formalnie sprawdzane podczas kompilacji.

Np. określając typ zmiennej X jako integer programista ustala, że ta zmienna ma przechowywać wartości całkowite. Dzięki temu możliwe jest sprawdzenie, czy wszystkie odwołania do tej zmiennej w programie mają kontekst, w jakim może być użyta wartość całkowita.

Mocna typologiczna kontrola poprawności programów okazała się cechą skutecznie eliminującą błędy popełniane przez programistów.

**7.7 Omów pojęcie i własności podstawowej jednostki działalności systemu - transakcji.**

Transakcje umożliwiają zachowanie spójności wielu jednocześnie działających procesów. „Ręczna” synchronizacja lub umawianie się są niepotrzebne.

Transakcje umożliwiają uniknięcie niespójności danych i przetwarzania związanych z dowolnymi awariami sprzętu, błędami w oprogramowaniu, niedyspozycją personelu, itd.

**7.8 Wymień typowe środowiska implementacji i krótko je omów.**

* **Środowiska języków proceduralnych**

Procesy i moduły wysokiego poziomu mogą odpowiadać całym aplikacjom.

Grupy procedur i funkcji - odpowiadają poszczególnym funkcjom systemu.

Składy/zbiorniki danych w projekcie odpowiadają strukturom danych języka lub

strukturom przechowywanym na pliku.

Języki proceduralne dają niewielkie możliwości ograniczenia dostępu do danych.

Poszczególne składowe struktur są dostępne wtedy, gdy dostępna jest cała struktur

Istnieje możliwość programowania w stylu obiektowym, ale brakuje udogodnień takich jak klasy, dziedziczenia, metod wirtualnych.

* **Środowiska języków obiektowych**

Bardzo przydatne jako środowisko implementacji projektu obiektowego, gdyż odwzorowanie pomiędzy modelem projektowym i implementacyjnym jest proste.

Z reguły jednak są niewystarczające w przypadku dużych zbiorów przetwarzanych danych i wymagają współpracy z bazą danych.

Większość języków obiektowych to języki hybrydowe, powstające w wyniku dołożenia cech obiektowości do języków proceduralnych. Najbardziej klasycznym przypadkiem takiego rozwiązania jest C++.

* **Środowiska relacyjnych baz danych**

W tej chwili są to najlepiej rozwinięte środowiska baz danych, pozwalające na zaimplementowanie systemów bazujących na dużych zbiorach danych.

**Plusy:**

wielodostęp

automatyczna weryfikacji więzów integralności

prawa dostępu dla poszczególnych użytkowników

wysoka niezawodność

rozszerzalność (ograniczona)

możliwość rozproszenia danych

dostęp na wysokim poziomie (SQL, ODBC, JDBC)

**Minusy**:

skomplikowane odwzorowanie modelu pojęciowego

mała efektywność dla pewnych zadań (kaskadowe złączenia)

ograniczenia w zakresie typów

brak hermetyzacji i innych cech obiektowości

zwiększenie długości kodu, który musi napisać programista

* **Środowiska obiektowych baz danych**

Zaletą modelu obiektowego baz danych jest wyższy poziom abstrakcji, który umożliwia zaprojektowanie i zaprogramowanie tej samej aplikacji w sposób bardziej skuteczny, konsekwentny i jednorodny.

Uproszczenie i usystematyzowanie procesu projektowania i programowania, minimalizacja liczby pojęć, zwiększenie poziomu abstrakcji, zmniejszenie dystansu pomiędzy fazami analizy, projektowania i programowania oraz zwiększeniu nacisku na rolę czynnika ludzkiego.

W stosunku do modelu relacyjnego, obiektowość wprowadza więcej pojęć, które wspomagają procesy myślowe zachodzące przy projektowaniu i implementacji.

Obiektowe bazy danych (ObjectStore, O2, Versant, Gemstone, Poet, Objectivity/DB, Jasmine, Jade, itd.) osiągają dojrzałość, ale nie pokonały jeszcze bariery nieufności powszechnego (dużego) klienta.

* **Środowiska obiektowo-relacyjnych BD**

Sukces obiektowości w zakresie ideologii i koncepcji spowodował wprowadzenie wielu cech obiektowości, takich jak klasy, metody, dziedziczenie, abstrakcyjne typy danych, do systemów relacyjnych. „Częściowa obiektowość” jest wprowadzona do większości systemów relacyjnych znajdujących się na rynku.

Takie podejście jest określane jako „hybrydowe” lub „obiektowo-relacyjne”. Ostatnio karierę robi także termin „uniwersalny serwer” (universal server), hasło marketingowe eksponujące możliwość zastosowania systemu do przechowywania i przetwarzania obiektów, relacji, danych multimedialnych, itd. Podstawą ideologiczną systemów obiektowo-relacyjnych jest zachowanie sprawdzonych technologii relacyjnych (np. SQL) i wprowadzanie na ich wierzchołku innych własności, w tym obiektowych.

* **Środowiska programów użytkowych**

Przykładem może być Microsoft Office, który można przystosować do różnych zastosowań. Np. cechy Microsoft Excel:

- Zawiera pełny proceduralny język Visula Basic dla Aplikacji

- Obejmuje szeroko rozbudowaną bibliotekę obiektów, udostęniającą praktycznie wszystkie możliwości pakietu.

- Pozwala na nagrywanie makrodefinicji w stylu Visual Basic.

- Posiada możliwość dialogowego projektowania interfejsu użytkownika: projektowanie dialogów, menu i pasków narzędziowych, umieszczanie pól dialogowych na arkuszach, definiowanie reakcji na zdarzenia

- Zawiera debugger ułatwiający uruchamianie programów

- Pozwala na dystrybucję aplikacji bez rozprowadzania kodu źródłowego

- Obejmuje rozbudowane możliwości współpracy ze standardami DLL, DDE, OLE, ODBC

**7.9 Czy faza implementacji może być automatyzowana ? Jeśli tak, to omów znane Ci sposoby, narzędzia i efekty automatyzacji.**

Programiści mogą bezpośrednio korzystać (przeglądać) diagramy i słownik danych korzystając z narzędzia CASE.

Niektóre systemy CASE (lower) dostarczają generatorów kodu, które generują programy lub ich szablony. Typowe elementy kodu:

 - skrypty tworzące relacje w bazie danych

 - definicje struktur danych

 - nagłówki procedur i funkcji

 - definicje klas

 - nagłówki metod

Kod jest uzupełniany wieloma komentarzami na podstawie informacji ze słownika danych. Niektóre narzędzia CASE umożliwiają interfejs do narzędzi RAD.

Jednym z przykładowych narzędzi jest vertabello które na podstawie stworzonego diagramu encji potrafi wygenerować skrypt tworzący baze danych oraz skrypt usuwający baze danych.

**7.10 Omów podstawowe rezultaty fazy implementacji.**

* Poprawiony dokument opisujący wymagania
* Poprawiony model analityczny
* Poprawiony projekt, który od tej pory stanowi już dokumentację techniczną
* Kod składający się z przetestowanych modułów
* Raport opisujący testy modułu
* Zaprojektowana i dostrojona baza danych
* Harmonogram fazy testowania

**8.1 Jakie zadania wykonywane są w fazie instalacji ?**

Szkolenie użytkowników końcowych i administratorów systemu

Instalacja sprzętu i przeniesienie oprogramowania

Wypełnienie baz danych

Nadzorowane korzystanie z systemu, często równoległe z tradycyjnym sposobem pracy

Usuwanie błędów w oprogramowaniu i dokumentacji użytkowej

Przekazanie systemu klientowi

**8.2 Jakim trudnościom w fazie instalacji musi stawić czoło kierownictwo przedsięwzięcia programistycznego?**

**Szkolenie użytkowników:** zaleca się, aby przeprowadzały je osoby, które były zaangażowane w prowadzenie przedsięwzięcia. Osobom tym będzie łatwiej nawiązać kontakt z przyszłymi użytkownikami.

**Wypełnienie bazy danych** jest często bardzo żmudnym procesem, wymagającym wprowadzenia danych z nośników papierowych. Niekiedy część danych jest w formie elektronicznej - wtedy z reguły potrzebne są specjalne programy konwersji. Konwersja jest łatwiejsza, jeżeli znana jest specyfikacja struktury starej BD.

**Ważne jest planowanie i harmonogramowanie prac**. W tej fazie pojawia się szereg problemów, np. konieczność usunięcia błędów i wprowadzenia modyfikacji. Z reguły, wykonawcy systemu nie mogą zarezerwować w pełni swojego czasu na prace związane z instalacją. Z drugiej strony, użytkownicy nie mogą zaniechać wykonywania przez nich bieżących prac.

**Pewien opór klienta przed zmianą sposobu pracy**. Często użytkownicy systemu są to osoby po raz pierwszy stykający się z systemem (inni niż ci, którzy uczestniczyli w poprzednich fazach). Ważne jest uzyskanie ich akceptacji.

**8.3 jakie zadania stoją przed kierownictwem przedsięwzięcia w fazie konserwacji?**

Wysoka jakość definicji wymagań, modelu i projektu

Dobra znajomość środowiska implementacji

Właściwa motywacja osób wykonujacych konserwację oprogramowania

Właściwe oszacowanie kosztów konserwacji

**8.4 Na czym polega inżynieria odwrotna? Kiedy i po co jest wykorzystywana?**

Narzędzie i metoda konserwacji oprogramowania.

Narzędzia inżynierii odwrotnej (reverse engineering). Są to narzędzia umożliwiające odtworzenie bardziej abstrakcyjnej postaci oprogramowania z postaci szczegółowej. Np. odtworzenie dokumentacji technicznej na podstawie kodu programu, odtworzenie źródłowego kodu programu na podstawie kodu skompilowanego (dekompilacja), odtworzeniu modelu logicznego bazy danych na podstawie jej fizycznej struktury, odtworzenie pojęciowego diagramu klas na podstawie deklaracji w języku programowania, itp.

**8.5 Omów rodzaje modyfikacji, jakie mogą mieć miejsce w fazie konserwacji i ich wpływ na koszty tej fazy.**

**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.

Analiza powinna uwzględniać:

Znaczenie wprowadzenia zmiany dla użytkowników

Koszt wprowadzenia zmiany

Wpływ zmiany na poszczególne składowe systemu

Wpływ zmiany na poszczególne składowe dokumentacji technicznej.

Dopiero po dokonaniu oceny zmiany podejmowana jest decyzja o jej ewentualnej realizacji. W przypadku bardzo dużych przedsięwzięć może zostać powołana w tym celu specjalna komisja.

**8.6 Jakie czynniki mogą mieć wpływ na wysokość kosztów konserwacji?**

Obiektywne czynniki wpływające na koszty konserwacji:

**Stabilność środowiska w którym pracuje system**. Zmiany zachodzące w przepisach prawnych, zmiany struktury organizacyjnej i sposobów działania po stronie klienta prowadzą do zmian wymagań wobec systemu.

**Stabilność platformy sprzętowej i oprogramowania systemowego**

**Czas użytkowania systemu**. Całkowite koszty konserwacji rosną, gdy system jest eksploatowany przez dłuższy czas.

**8.7 Omów sposoby i środki służące zmniejszaniu kosztów konserwacji.**

Wiele działań zmierzających do redukcji kosztów konserwacji musi być podjęte już w fazie budowy systemu.

**Znajomość dziedziny problemu**. Jeżeli analitycy pracujący nad systemem dobrze znają daną dziedzinę problemu, mają mniej trudności z właściwym zebraniem wymagań oraz budową oddającego rzeczywistość modelu.

**Wysoka jakość modelu i projektu**, w szczególności jego spójność, stopień powiązania składowych oraz przejrzystość.

**Wysoka jakość dokumentacji technicznej**. Powinna ona: w pełni odpowiadać systemowi , być wystarczająco szczegółowa , być zgodna z przyjętymi w firmie standardami.

**Stabilność personelu**. Niezależnie od jakości dokumentacji, pewne aspekty systemu są znane tylko osobom bezpośrednio uczestniczącym w realizacji. Niekoniecznie muszą one same dokonywać modyfikacji, ale mogą istotnie wspomagać konsultacjami.

**Środowisko implementacji**. Zaawansowane środowisko implementacji sprzyja skróceniu czasu niezbędnego na wprowadzenie modyfikacji.

**Niezawodność oprogramowania**. Wysoka niezawodność oprogramowania przekazanego klientowi zmniejsza liczbę modyfikacji.

**Inżynieria odwrotna**. Pod tym pojęciem rozumie się odtwarzanie dokumentacji technicznej na podstawie istniejącego oprogramowania.

**Zarządzanie wersjami.**

**8.8 Wymień i krótko omów typowe moduły wchodzące w skład narzędzi CASE.**

**Edytory notacji graficznych** Tworzenie i edycja diagramów wykorzystywanych w fazach określania wymagań.

Tworzenie i edycja powiązań pomiędzy poszczególnymi symbolami i diagramami oraz nawigowanie po sieci powiązanych diagramów. Wydruk diagramów.

**Repozytorium narzędzia CASE (słownik)** Jest to baza danych o realizowanym projekcie oraz narzędzia służące do jej edycji i przeglądania.

- Wprowadzenie oraz edycja specyfikacji modelu i projektu, a także innych informacji związanych z przedsięwzięciem.

- Wyszukiwanie pożądanej informacji

**Moduł kontroli poprawności** Pewne błędy mogą być wykrywane na bieżąco w trakcie edycji diagramów i słownika danych., np. uczynienie klasy swoja własną specjalizacją.

**Moduł kontroli jakości** Pewne systemy pozwalają na automatyczną ocenę pewnych miar jakości projektu. Dotyczy to szczególnie złożoności oraz stopnia powiązania składowych.

**Generator raportów** Służy do przygotowania raportów na podstawie zawartości słownika danych. Niektóre raporty są parametryczne. Narzędzia CASE zawierają sporo gotowych generatorów raportów. Niektóre z nich pozwalają na definiowanie własnych.

**Generator dokumentacji technicznej** Moduł służący do przygotowania dokumentacji technicznej złożonej z szeregu diagramów. Swobodne formatowanie dokumentów. Przykładowe dokumenty. Łatwe i efektywne uaktualnienie dokumentacji po dokonaniu zmian w projekcie

**Generatory kodu** Narzędzia służące do generacji kodu w rozmaitych językach programowania. Często generują szkielety, które muszą być uzupełnione przez użytkownika dodatkowym kodem. Wygenerowany kod jest uzupełniony o komentarze i inne informacje. Może także zawierać pewne elementy do modyfikacji. Nazwy użyte w projekcie przechodzą do wynikowego kodu (ewentualnie są skracane).

**Moduł zarządzania wersjami** Umożliwia kontrolę różnych wersji projektu powstających ze względu na konieczność grupowego wprowadzania zmian oraz wskutek wielu środowisk informatycznych (sprzęt i oprogramowania) oraz różnych zastosowań.

**Moduł projektowania interfejsu użytkownika** Dotyczy projektowania dialogów, okien, menu. Zaletą jest wykorzystanie informacji znajdujących się w słowniku danych. Pozwala to np. na automatyczne wygenerowanie dialogu do edycji pewnej struktury danych.

Integracja z RAD. **Moduł inżynierii odwrotnej**

**8.9 Oceń efektywność narzędzi CASE. Co jest powodem niskiej jak dotąd efektywności narzędzi CASE? Wymień kilka znanych ci narzędzi tego typu.**

**Traktowanie narzędzi CASE wyłącznie jako generatorów kodu.** Nie jest to efektywne przy braku rzetelnego podejścia do analizy i projektowania: • nakłady na implementację stanowią tylko ok. 15-30% całych nakładów • koszt błędów popełnionych w fazie implementacji jest stosunkowo niewielki • istnieją inne, tańsze narzędzia programistyczne (RAD)

**Nieznajomość metodyki analizy i projektowania**. Narzędzia CASE nie zwalniają z myślenia, wiedzy i doświadczenia.

**Niewłaściwa organizacja i zarządzanie przedsięwzięciem**. Nieuporządkowanie prac, brak planu, brak właściwych ocen, brak monitorowania postępu, itd.

**Zbyt wysokie oczekiwania w stosunku do narzędzia CASE**. Może ono zredukować koszty co najwyżej o 50%, koszt wdrożenia jest wysoki, efekty pojawiają się z pewnym opóźnieniem, wymaga dyscypliny w przedsięwzięciu

Narzędzie CASE nie przesądza w auto-magiczny sposób o powodzeniu projektu. Zastosowanie obiektowego CASE niekoniecznie oznacza “nowoczesność” projektu. Stosowanie narzędzi CASE często przynosi znikome efekty.

Np. Oracle CASE, EasyCASE, CASE 4.0, ObjectiF, Select OMT Professional, System Architect, ObjectTeam, Paradigm Plus, Rational Rose, Select Enterprise

**8.10 Na czym polega różnica pomiędzy Upper- a Lower-CASE? Oceń przydatność obu narzędzi CASE w realizacji przedsięwzięcia programistycznego.**

**Upper-CASE**: wspomaganie wczesnych faz prac nad oprogramowaniem, w szczególności fazy analizy (potrzeby analityków i projektantów). Narzędzia te nie są związane z konkretnym środowiskiem implementacyjnym.

**Lower-CASE**: wspomaganie faz projektowania i implementacji (potrzeby programistów). Narzędzia te są reguły ściśle związane z konkretnym środowiskiem implementacji.

**9.1 Co rozumiesz pod pojęciami: atestowanie i weryfikacja? Krótko omów definicje i różnice między tymi pojęciami.**

**Weryfikacja** - testowanie zgodności systemu z wymaganiami zdefiniowanymi w fazie określenia wymagań.

**Atestowanie** - ocena systemu lub komponentu podczas lub na końcu procesu jego rozwoju na zgodności z wyspecyfikowanymi wymaganiami. Atestowanie jest więc weryfikacją końcową.

**9.2 Co to jest weryfikacja? Wymień i krótko omów: co podlega weryfikacji oraz jakie znasz związane z nią czynności.**

Proces weryfikacji oprogramowania można określić jako poszukiwanie i usuwanie błędów na podstawie obserwacji błędnych wykonań oraz innych testów.

**Weryfikacja oznacza**: 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.

**9.3 Co to jest audyt. Czy audyt oprogramowania jest potrzebny.**

Audytem nazywany jest niezależny przegląd i ocena jakości oprogramowania, która zapewnia zgodność z wymaganiami na oprogramowanie, a także ze specyfikacją, generalnymi założeniami, standardami, procedurami, instrukcjami, kodami oraz kontraktowymi i licencyjnymi wymaganiami.

**9.4 Omów pojęcie inspekcji. Jaki jest jej przebieg.**

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

**Inicjowanie** zgłoszenie potrzeby inspekcji; wyłonienie lidera inspekcji

**Planowanie** lider ustala uczestników, listy kontrolne, zbiory reguł, tempo kontroli, daty spotkań kontrolnych

**Spotkanie inicjujące** ustalenie ról, ustalenie celów i oczekiwań, dystrybucja dokumentu, szkolenie w inspekcjach

**Kontrola indywidualna** uczestnicy sprawdzają dokument względem zadanych kryteriów, reguł i list kontrolnych (znaleźć jak najwięcej unikalnych błędów)

**Spotkanie kontrolne** Notowanie uwag z kontroli indywidualnej; Każda uwaga jest kwalifikowana jako „zagadnienie” (potencjalny błąd), „pytanie o intencję”, „propozycja poprawy procesu”; Szukanie nowych zagadnień; Poprawa procesu inspekcji.

**Poprawa produktu:** edytor (najczęściej autor) rozwiązuje zagadnienia; prawdziwy problem może być inny niż jest to zgłoszone; dokument jest redagowany by uniknąć błędnych interpretacji

**Kontynuacja**: lider sprawdza, że obsłużono wszystkie zagadnienia: są poprawione lub są w systemie zarządzania konfiguracją; sprawdza kompletność a nie poprawność

**Decyzja o gotowości**: lider podejmuje decyzję czy produkt jest gotowy do przekazania dalej (np. liczba błędów w określonym limicie)

**Rozpowszechnienie dokumentu**

**9.5 Porównaj pojęcia: błąd i błędne wykonanie.**

**Błąd** - niepoprawna konstrukcja znajdująca się w programie, która może doprowadzić do niewłaściwego działania. **Błędne wykonanie** - niepoprawne działanie systemu w trakcie jego pracy.

Błąd może prowadzić do różnych błędnych wykonań.

To samo błędne wykonanie może być spowodowane różnymi błędami.

**9.6 Jakie testy traktują system jako czarną skrzynkę. Omów ten rodzaj testów.**

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.

Testujący powinni podzielić dane wejściowe w „klasy równoważności”, co do których istnieje duże przypuszczenie, że będą produkować te same błędy. „Klasy równoważności” mogą być również zależne od wyników zwracanych przez testowane funkcje.

Testy dynamiczne, funkcjonalne

**9.7 Jakie testy traktują system jak białą skrzynkę? Omów ten rodzaj testów.**

Tak określa się sprawdzanie wewnętrznej logiki oprogramowania. Testowanie n/z białej skrzynki pozwala sprawdzić wewnętrzną logikę programów poprzez odpowiedni dobór danych wejściowych, dzięki czemu można prześledzić wszystkie ścieżki przebiegu sterowania programu. Tradycyjnie programiści wstawiają kod diagnostyczny do programu aby śledzić wewnętrzne przetwarzanie. Debuggery pozwalają programistom obserwować wykonanie programu krok po kroku. Często niezbędne staje się wcześniejsze przygotowanie danych testowych lub specjalnych programów usprawniających testowanie (np. programu wywołującego testowaną procedurę z różnymi parametrami), powinny być dobrane w taki sposób, aby każda ścieżka w programie była co najmniej raz przetestowana.

Ograniczeniem testowania jest niemożliwość pokazania brakujących funkcji w programie.

Testy dynamiczne, strukturalne

**9.8 Omów typowe fazy testowania systemu.**

**Testowanie modułów** - Są one wykonywane już w fazie implementacji bezpośrednio po zakończeniu realizacji poszczególnych modułów

**Testowanie Integracji**

**Testowanie całości systemu** - tej fazie integrowane są poszczególne moduły i testowane są poszczególne podsystemy oraz system jako całość

**Testowanie akceptacji użytkowników** - W przypadku oprogramowania realizowanego na zamówienie system przekazywany jest do przetestowania przyszłemu użytkownikowi. Testy takie nazywa się wtedy testami alfa. W przypadku oprogramowania sprzedawanego rynkowo testy takie polegają na nieodpłatnym przekazaniu pewnej liczby kopii systemu grupie użytkowników. Testy takie nazywa się testami beta.

**9.9 Co podlega testowaniu? Krótko omów elementy podlegające testowaniu.**

**Wydajność systemu** i poszczególnych jego funkcji (czy jest satysfakcjonująca).

**Interfejsy systemu** na zgodność z wymaganiami określonymi przez użytkowników

**Własności operacyjne** systemu, np. wymagania logistyczne, organizacyjne, użyteczność/ stopień skomplikowania instrukcji kierowanych do systemu, czytelność ekranów, operacje wymagające zbyt wielu kroków, jakość komunikatów systemu, jakość informacji o błędach, jakość pomocy.

**Testy zużycia zasobów**: zużycie czasu jednostki centralnej, zużycie pamięci operacyjnej, przestrzeni dyskowej, itd. **Zabezpieczenie systemu:** odporność systemu na naruszenia prywatności, tajności, integralności, spójności i dostępności. Testy powinny np. obejmować: - zabezpieczenie haseł użytkowników - testy zamykania zasobów przed niepowołanym dostępem - testy dostępu do plików przez niepowołanych użytkowników - testy na możliwość zablokowania systemu przez niepowołane osoby

**Przenaszalność oprogramowania**: czy oprogramowanie będzie działać w zróżnicowanym środowisku (np. różnych wersjach Windows 95, NT, Unix), przy różnych wersjach instalacyjnych, rozmiarach zasobów, kartach graficznych, rozdzielczości ekranów, oprogramowaniu wspomagającym (bibliotekach), ...

**Niezawodność oprogramowania**, zwykle mierzoną średnim czasem pomiędzy błędami.

**Odtwarzalność oprogramowania** (maintainability), mierzoną zwykle średnim czasem reperowania oprogramowania po jego awarii. Pomiar powinien uwzględniać średni czas od zgłoszenia awarii do ponownego sprawnego działania.

Bezpieczeństwo oprogramowania: stopień minimalizacji katastrofalnych skutków wynikających z niesprawnego działania. (Przykładem jest wyłączenie prądu podczas działania w banku i obserwacja, co się w takim przypadku stanie.)

**Kompletność i jakość założonych funkcji systemu.**

**Nieprzekraczanie ograniczeń**, np. na zajmowaną pamięć, obciążenia procesora

**Modyfikowalność oprogramowania**, czyli zdolność jego do zmiany przy zmieniających się założeniach lub wymaganiach

**Obciążalność oprogramowania**, tj. jego zdolność do poprawnej pracy przy ekstremalnie dużych obciążeniach. Np. maksymalnej liczbie użytkowników, bardzo dużych rozmiarach plików, dużej liczbie danych w bazie danych, ogromnych (maksymalnych) zapisach, bardzo długich liniach danych źródłowych. W tych testach czas nie odgrywa roli, chodzi wyłącznie o to, czy system poradzi sobie z ekstremalnymi rozmiarami danych lub ich komponentów oraz z maksymalnymi obciążeniami na jego wejściu.

**Skalowalność** **systemu**, tj. spełnienie warunków (m.in. czasowych) przy znacznym wzroście obciążenia.

**Akceptowalność systemu**, tj. stopień usatysfakcjonowania użytkowników.

**Jakość dokumentacji**, pomocy, materiałów szkoleniowych, zmniejszenia bariery dla nowicjuszy.

**9.10 Przedstaw schemat przeprowadzania 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.

Powyższe czynności powtarzane są cyklicznie

**10.1 Określ sposoby szacowania niezawodności oprogramowania i omów do czego służą.**

**Prawdopodobieństwo błędnego wykonania** podczas realizacji transakcji. Każde błędne wykonanie powoduje zerwanie całej transakcji. Miarą jest częstość występowania transakcji, które nie powiodły się wskutek błędów.

**Częstotliwość występowania błędnych wykonań**: ilość błędów w jednostce czasu. Np. 0.1/h oznacza, że w ciągu godziny ilość spodziewanych błędnych wykonań wynosi 0.1. Miara ta jest stosowana w przypadku systemów, które nie mają charakteru transakcyjnego.

**Średni czas między błędnymi wykonaniami** - odwrotność poprzedniej miary.

**Dostępność**: prawdopodobieństwo, że w danej chwili system będzie dostępny do użytkowania. Miarę tę można oszacować na podstawie stosunku czasu, w którym system jest dostępny, do czasu od wystąpienia błędu do powrotu do normalnej sytuacji. Miara zależy nie tylko od błędnych wykonań, ale także od narzutu błędów na niedostępność systemu.

**10.2 Porównaj testy strukturalne z testami funkcjonalnymi.**

**Testy funkcjonalne** które zakładają znajomość jedynie wymagań wobec testowanej funkcji. System jest traktowany jako czarna skrzynka, która w nieznany sposób realizuje wykonywane funkcje. Testy powinny wykonywać osoby, które nie były zaangażowane w realizację testowanych fragmentów systemu.

**Testy strukturalne** które zakładają znajomość sposobu implementacji testowanych funkcji

W przypadku testów strukturalnych, dane wejściowe dobiera się na podstawie analizy struktury programu realizującego testowane funkcje. **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.

**10.3 Wyjaśnij pojęcie testów dynamicznych. Jakie błędy są przez nie wykrywane? Jaki rodzaj testów dynamicznych jest szczególnie efektywny?**

Testy dynamiczne, polegają na wykonywaniu (fragmentów) programu i porównywaniu uzyskanych wyników z wynikami poprawnymi.

Dynamiczne testy zorientowane na wykrywanie błędów dzieli się na: Testy funkcjonalne i Testy strukturalne

**10.4 Wyjaśnij pojęcie testów statycznych. Jakie błędy są przez nie wykrywane? Jaki rodzaj testów statycznych jest szczególnie efektywny.**

Polegają na analizie kodu bez uruchomienia programu. Techniki są następujące:

• dowody poprawności • metody nieformalne

Statyczne metody nieformalne polegają na analizie kodu przez programistów. Dwa niewykluczające się podejścia:

• śledzenie przebiegu programu (wykonywanie programu “w myśli” przez analizujące osoby)

• wyszukiwanie typowych błędów

Błędy

 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

**10.5 Omów pojęcie eksplozji danych testowych.**

Jeżeli danych testowych (wartości typowe, wartości graniczne) jest wiele, to mamy do czynienia z kombinatoryczną eksplozją przypadków testowych. Dzieląc dane wejściowe na klasy należy więc brać pod uwagę rozmaite kombinacje elementarnych warunków. W praktyce przetestowanie wszystkich kombinacji danych wejściowych jest najczęściej niemożliwe.

**10.6 Według jakich kryteriów dobiera się dane testowe w przypadku testów strukturalnych.**

**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.

**10.7 Porównaj programy uruchamiające i analizatory przykrycia kodu. Co czego służą i jakie powinny mieć własności.**

**Programy uruchamiające** (*debuggers*) Mogą być przydatne dla wewnętrznego testowania jak i dla testowania przez osoby zewnętrzne. Zakładają testowanie na zasadzie białej skrzynki (znajomość kodu).

**Analizatory przykrycia kodu** *(coverage analysers)*Są to programy umożliwiające ustalenie obszarów kodu źródłowego, które były wykonane w danym przebiegu testowania. Umożliwiają wykrycie martwego kodu, kodu uruchamianego przy bardzo specyficznych danych wejściowych oraz (niekiedy) kodu wykonywanego bardzo często (co może być przyczyną wąskiego gardła w programie).

**10.8 Omów technikę posiewania błędów. Kiedy jest efektywna?**

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. Technika ta pozwala również na przetestowanie skuteczności metod testowania

**10.9 Porównaj testowanie wstępujące i zstępujące.**

**Testowanie wstępujące**: najpierw testowane są pojedyncze moduły, następnie moduły coraz wyższego poziomu, aż do osiągnięcia poziomu całego systemu. Zastosowanie tej metody nie zawsze jest możliwe, gdyż często moduły są od siebie zależne. Niekiedy moduły współpracujące można zastąpić implementacjami szkieletowymi.

**Testowanie zstępujące**: rozpoczyna się od testowania modułów wyższego poziomu. Moduły niższego poziomu zastępuje się implementacjami szkieletowymi. Po przetestowaniu modułów wyższego poziomu dołączane są moduły niższego poziomu. Proces ten jest kontynuowany aż do zintegrowania i przetestowania całego systemu.

**10.10 Porównaj (definiując pojęcie i cele testowania) testy obciążeniowe i testy odporności.**

**Testy obciążeniowe** 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** Celem tych testów jest sprawdzenie działania w przypadku zajścia niepożądanych zdarzeń, np. zaniku zasilania

**11.1 Krótko omów co rozumiesz pod pojęciem jakość oprogramowania.**

- 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.

- Podstawą obiektywnych wniosków co do jakości oprogramowania są pomiary pewnych parametrów użytkowych (niezawodności, szybkości, itd.) w realnym środowisku, np. przy użyciu metod statystycznych.

- Niestety, obiektywne pomiary cech produktów programistycznych są utrudnione lub niemożliwe. Jakość gotowych produktów programistycznych jest bardzo trudna do zmierzenia ze względu na ich złożoność (eksplozja danych testowych), wieloaspektowość, identyczność wszystkich kopii produktu, oraz niską przewidywalności wszystkich aspektów ich zastosowań w długim czasie

**11.2** **Czy łatwo jest zmierzyć jakość oprogramowania? Jakie czynniki stanowią przeszkodę w ocenie jakości**

- Oceny jakości najczęściej muszą być znane zanim powstanie gotowy, działający produkt, co wyklucza zastosowanie obiektywnych metod pomiarowych.

- Wiele czynników składających się na jakość produktu jest niemierzalna.

- Produkty programistyczne są złożone i wieloaspektowe, co powoduje trudności w wyodrębnieniu cech mierzalnych, które odzwierciedlałyby istotne aspekty jakości.

- Produkty programistyczne mogą działać w różnych zastosowaniach, o różnej skali. Pomiary jakości mogą okazać się nieadekwatne przy zmianie skali (np. zwiększonej liczbie danych lub użytkowników), w innym środowisku, itp.

- Pomiary mogą okazać się bardzo kosztowne, czasochłonne lub niewykonalne (z powodu niemożliwości stworzenia środowiska pomiarowego przed wdrożeniem);

- Nie ma zgody co do tego, w jaki sposób pomierzone cechy danego produktu składają się na syntetyczny wskaźnik jego jakości.

- Stąd oceny jakości produktów programistycznych są skazane na metody spekulacyjne, oparte na uproszczeniach oraz dodatkowych założeniach, algorytmach, wzorach i heurystykach.

**11.3 Omów koncepcję TQM – zarządzania przez jakość**

Koncepcja wynaleziona przez Japończyka Eiji Toyodę dla potrzeb naprawy japońskiego przemysłu motoryzacyjnego - 1950 r. Główna jej myśl mówiła o tym, że w związku z tym, że to klient stanowi o rentowności przedsiębiorstwa, to należy tak sterować wszystkimi fazami procesu produkcyjnego wyrobu, aby klient był zadowolony z jakości tego wyrobu, TQM została sformalizowana przez Amerykanów. Wszystkie one obracają się jednak wokół zasady Toyody: „Jakość jest najważniejszym kryterium oceny przydatności produktów dla klienta, a to właśnie klient umożliwia funkcjonowanie wytwórcy tych produktów”. Stąd wniosek, że producent wytwarzający produkty kiepskie powinien wypaść z rynku.

**11.4 Omów (zgodnie z terminologią ISO 9000) pojęcia jakość, system jakości**

**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

**11.5 Omów (zgodnie z terminologią ISO 9000) pojęcia zarządzanie jakością, polityka jakości**

**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

**11.6 Omów (zgodnie z terminologią ISO 9000) pojęcia audyt oprogramowania, audyt 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

**11.7 Omów na czym polega ZJO (Zapewnienia Jakości Oprogramowania)**

Zgodnie z normą jest to „planowany i systematyczny wzorzec wszystkich działań potrzebnych dla dostarczenia adekwatnego potwierdzenia że element lub produkt jest zgodny z ustanowionymi wymaganiami technicznymi”.

**ZJO** oznacza sprawdzanie:

czy plany są zdefiniowane zgodnie ze standardami;

czy procedury są wykonywane zgodnie z planami;

czy produkty są implementowane zgodnie z planami.

Kompletne sprawdzenie jest zwykle niemożliwe. Projekty bardziej odpowiedzialne powinny być dokładniej sprawdzane odnośnie jakości. W ramach ZJO musi być ustalony plan ustalający czynności sprawdzające przeprowadzane w poszczególnych fazach projektu.

**11.8 Omów jakie procesy są obsługiwane przez personel ZJO (Zapewnienia Jakości Oprogramowania)**

**Tworzenie technologii** –tworzenie standardu –wdrażanie standardu

**Kontrola jakości** –ocena produktu –ocena procesu –zatwierdzanie jakości

**Analiza działalności firmy** –zbieranie danych –analiza danych

**Administrowanie siecią komputerową**

**Zarządzanie personelem**

**11.9 Jakie znasz normy dotyczące jakości; czego w kontekście jakości one dotyczą?**

Oprogramowanie jest rozumiane jako jeden z rodzajów wyrobów

ISO 8402 Terminologia

ISO/IEC 1508 Bezpieczeństwo oprogramowania systemów krytycznych

IEC/TC 56 Niezawodność oprogramowania systemów krytycznych

ISO 9000 Wytyczne wyboru modelu

ISO 9001 ISO 9002 ISO 9003 Modele systemu jakości

ISO 9004 Elementy systemu jakości

IEEE-730 podaje ogólne ramy planu zapewniania jakości

IEEE-983 uzupełnia i uszczegóławia IEEE-730

**11.10 Omów pojęcie CMM (model dojrzałości procesu wytwórczego)**

Wykorzystywany w procedurach klasyfikacji potencjalnych wykonawców oprogramowania dla Departamentu Obrony USA

Wyróżniono 5 poziomów dojrzałości wytwórców (poczynając od poziomu najniższego):

**poziom początkowy** - 1 (proces chaotyczny)

**poziom powtarzalny** - 2 (proces zindywidualizowany)

**poziom zdefiniowany** - 3 (proces zinstytucjonalizowany)

**poziom zarządzany** - 4 (proces + informacje zwrotne dla sterowania procesem)

**poziom optymalizujący** - 5 (proces + informacje zwrotne wpływające na ulepszenie procesu

Niewiele firm uzyskało poziom 3-cis

**12.1 Czym różni się metryka od pomiaru? Wyjaśnij oba pojęcia.**

**Pomiar** jest to proces, w którym atrybutom świata rzeczywistego przydzielane są liczby lub symbole w taki sposób, aby charakteryzować te atrybuty według jasno określonych zasad. Jednostki przydzielane atrybutom nazywane są miarą danego atrybutu.

**Metryka** jest to proponowana (postulowana) miara. Nie zawsze charakteryzuje ona w sposób obiektywny dany atrybut. Np. ilość linii kodu (LOC) jest metryką charakteryzującą atrybut “długość programu źródłowego”, ale nie jest miarą ani złożoności ani rozmiaru programu (choć występuje w tej roli).

112. **Jakie elementy oprogramowanie można pomierzyć? Podaj przykłady.**

* Proces: każde określone działanie w ramach projektu, wytwarzania lub eksploatacji oprogramowania.
* Produkt: każdy przedmiot powstały w wyniku procesu: kod źródłowy, specyfikację projektową, udokumentowaną modyfikację, plan testów, dokumentację, itd.
* Zasób: każdy element niezbędny do realizacji procesu: osoby,narzędzia, metody wytwarzania, itd.

113. **Jakie cechy zasobów mogą mierzalne bezpośrednio? Podaj wskaźniki syntetyczne dla atrybutów bezpośrednio mierzalnych.**

Atrybuty bezpośrednio mierzalne: wiek, cena, wielkość, poziom komunikacji, struktura, szybkość, wielkość pamięci, temperatura, oświetlenie.

Wskaźniki syntetyczne: wydajność, doświadczenie, inteligencja,  jakość, używalność, niezawodność, wygoda

114. **Jakie cechy produktów mogą mierzalne bezpośrednio? Podaj wskaźniki syntetyczne dla atrybutów bezpośrednio mierzalnych.**

 Atrybuty bezpośrednio mierzalne: rozmiar, ponowne użycie, modularność, nadmiarowość, funkcjonalność, poprawność składniową, rozmiar, spójność, złożoność, poziom pokrycia

 Wskaźniki syntetyczne: zrozumiałość, pielęgnacyjność, jakość, złożoność, niezawodność, używalność

115. **Jakie cechy procesów mogą mierzalne bezpośrednio? Podaj wskaźniki syntetyczne dla atrybutów bezpośrednio mierzalnych.**

 Atrybuty bezpośrednio mierzalne: czas, nakład pracy, liczba zmian wymagań, liczba znalezionych usterek specyfikacji, liczba znalezionych błędów kodu

 Wskaźniki syntetyczne:  jakość, koszt, stabilność, opłacalność

116. **Omów cechę złożoności w planowaniu projektu.**

 **Oprogramowanie**: decyzje strategiczne, analiza, projektowanie, konstrukcja, dokumentacja, wdrożenie, szkolenie, eksploatacja, pielęgnacja, modyfikacja

 **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

117. Oszacuj koszty związane z kolejnymi fazami wytwarzania oprogramowania (proporcjonalnie). Omów przedstawione oszacowanie.

118. **Do czego służy metoda COCOMO i jakie są jej wady?**

 COCOMO jest oparte na kilku formułach pozwalających oszacować całkowity koszt przedsięwzięcia na podstawie oszacowanej liczby linii kodu.

Jest to główna słabość tej metody, gdyż:

* liczba ta staje się przewidywalna dopiero wtedy, gdy kończy się faza projektowania architektury systemu; jest to za późno;
* pojęcie “linii kodu” zależy od języka programowania i przyjętych konwencji;
* pojęcie “linii kodu” nie ma zastosowania do nowoczesnych technik programistycznych, np. programowania wizyjnego.

119. **Omów na czym polega metoda punktów funkcyjnych i z jakich danych wejściowych korzysta się w tej metodzie?**

 Metoda punktów funkcyjnych oszacowuje koszt projektu na podstawie funkcji użytkowych, które system ma realizować. Stąd wynika, ze metoda ta może być stosowana dopiero wtedy, gdy funkcje te są z grubsza znane.

Metoda jest oparta na zliczaniu ilości wejść i wyjść systemu, miejsc przechowywania danych i innych kryteriów. Te dane są następnie mnożone przez zadane z góry wagi i sumowane. Rezultatem jest liczba „punktów funkcyjnych”.

120. **Omów podstawowe zadania kierownictwa przedsięwzięcia programistycznego.**

* Opracowanie propozycji dotyczących sposobu prowadzenia przedsięwzięcia
* Kosztorysowanie przedsięwzięcia
* Planowanie i harmonogramowanie przedsięwzięcia
* Monitorowanie i kontrolowanie realizacji przedsięwzięcia
* Dobór i ocena personelu
* Opracowanie i prezentowanie sprawozdań dla kierownictwa wyższego szczebla

121. **Omów wady i zalety typowych sposobów oceniania pracowników firmy programistycznej.**

 Testy osobowościowe - stosowanie ich wiąże się z następującymi trudnościami:

* Osobowość ludzka ma charakter dynamiczny (zmienia się). Wieloletnia praktyka zawodowa nie pozostaje bez wpływu na osobowość. Część cech osobowości może być nabyta i nie da się odkryć wstępnymi testami.
* Różne zadania mogą wymagać różnych cech osobowości. Inne powinien posiadać analityk (kontakt z klientem), inne zaś programista lub osoba testująca oprogramowanie. Ponadto, metody inżynierii oprogramowania ulegają zmianie, co pociąga za sobą inny stosunek pożądanych cech osobowości do aktualnych zadań.
* Osoby poddane testom będą starały się raczej odgadnąć pożądaną przez testujących odpowiedź niż odpowiadać zgodnie ze stanem faktycznym. Test nie będzie więc odzwierciedlał cech osobowości osoby, lecz raczej to, jak ta osoba wyobraża sobie cele i kryteria testowania oraz cechy pożądane przez pracodawcę.

122. **Jakimi twoim zdaniem powinien charakteryzować się dobry inżynier oprogramowania? Omów te cechy.**

 Umiejętność pracy w stresie. W pracy często zdarzają się okresy wymagające szybkiego wykonania złożonych zadań. Dla większości osób niewielki stres działa mobilizująco. Po przekroczeniu jednak pewnego progu następuje spadek możliwości danej osoby. Próg ten jest różny dla różnych osób.

 Zdolności adaptacyjne. Informatyka jest jedną z najszybciej zmieniających się dziedzin. Ocenia się, że 7-9 miesięcy przynosi w informatyce zmiany, które w innych dziedzinach zajmują 5-7 lat. Oznacza to konieczność stałego kształcenia dla wszystkich inżynierów oprogramowania - stałe poznawanie nowych narzędzi, sprzętu, oprogramowania, technologii, metod, sposobów pracy. Niestety, nie wszyscy to tempo wytrzymują. (Uśpienie, zajmowanie się jednym problemem w jednym środowisku przez lata jest w informatyce bardzo groźne!)

123. **Omów pojęcia wiedza semantyczna i wiedza składnikowa. Jaki według Ciebie powinny mieć wpływ na politykę kadrową?**

 Wiedza składniowa. Polega na mechanicznym zapamiętaniu pewnych faktów, bez ich istotnego przetworzenia. Jest słabo zintegrowana z wcześniej zdobytą wiedzą. Np. do takiej wiedzy zaliczamy reguły składniowe danego języka programowania.

Wiedza semantyczna (znaczeniowa). Fakty są zapamiętane nie w postaci ich formy, lecz w postaci znaczenia. Np. znajomość zasady instrukcji while, znajomość koncepcji pojęcia klasy i dziedziczenia, itd. Nowa wiedza jest zintegrowana z wcześniej zdobytą wiedzą.

Istnienie tych dwóch rodzajów wiedzy może mieć wpływ na politykę kadrową. Np. pracownik rozumiejący zasady obiektowości (wiedza semantyczna) może lepiej sobie poradzić niż pracownik dobrze znający składnię i reguły użycia poszczególnych konstrukcji C++ (wiedza składniowa).

Firmy przywiązują zbyt wielką wagę do wiedzy składniowej, np. znajomości konkretnych języków i systemów. W istocie, ta wiedza może być stosunkowo szybko nabyta (kilka tygodni przeciętnie na opanowanie nowego języka). Natomiast wiedza semantyczne może być przedmiotem lat studiów i doświadczeń.

124. **Omów podstawowe typy psychologiczne, pod kątem nastawienia do pracy w zespole. Jak dobierałbyś pracowników, wiedząc do jakich typów psychologicznych należą.**

* Zorientowani na zadania (task-oriented). Osoby samowystarczalne, zdolne, zamknięte, agresywne, lubiące współzawodnictwo, niezależne.
* Zorientowani na siebie (self-oriented). Osoby niezgodne, dogmatyczne, agresywne, zamknięte, lubiące współzawodnictwo, zazdrosne.
* Zorientowani na interakcję (interaction-oriented). Osoby nieagresywne, o niewielkiej potrzebie autonomii i indywidualnych osiągnięć, pomocne, przyjazne.

Osoby typu 1 są efektywne, o ile pracują w pojedynkę. Zespół złożony z takich osób może być jednak nieefektywny. Lepsze wyniki dają zespoły złożone z typów 3. Typ 1 i 2 może być także efektywny w zespole, o ile jest odpowiednio motywowany przez kierownictwo. Typy 3 są konieczne w fazie wstępnej wymagającej intensywnej interakcji z klientem.

125. **Wymień (z omówieniem funkcji, jakie pełnią) osoby wchodzące w skład zespołu konstruującego produkt programistyczny.**

* Kierownik programu/przedsięwzięcia
* Analityk - osoba bezpośrednio kontaktująca się z klientem, której celem jest określenie wymagań i budowa modelu systemu
* Projektant - osoba odpowiedzialna za realizację oprogramowania. Może
* posiadać bardziej wyspecjalizowane funkcje:
	+ Projektant interfejsu użytkownika
	+ Projektant bazy danych
* Programista - osoba implementująca oprogramowanie
* Osoba wykonująca testy
* Osoba odpowiedzialna za konserwację oprogramowania
* Ekspert metodyczny - osoba szczególnie dobrze znająca stosowaną metodykę
* Ekspert techniczny - osoba szczególnie dobrze znająca sprzęt i narzędzia

126. Jak rozumiesz pojęcie zapewnienie jakości? Porównaj je z pojęciem testowanie oprogramowania.

 “Zapewnienie jakości to zestaw procedur, technik i narzędzi mających zapewnić, że tworzony produkt spełnia narzucone standardy. Jeżeli standardy nie są jawnie określone, zapewnienie jakości oznacza zaspokojenie minimalnych, rynkowych wymagań jakości.” (Bersoff, 1984)

**127. Porównaj dwa typy struktury zespołu programistycznego: sieciowa i gwiaździsta.**

 

**sieciowa** - zalety:

* Dzięki ścisłej współpracy członkowie zespołu wzajemnie kontrolują swoją pracę.

Szybko osiągane są standardy jakości.

* Umożliwia realizację idei wspólnego programowania
* Ponieważ praca członków zespołu jest znana dla innych członków, łatwo mogą oni przejąć obowiązki pracownika, który opuścił zespół.

Struktura sieciowa nie może liczyć więcej niż 8 osób.

**gwiaździsta**:
Jest przydatna wtedy, gdy w skład zespołu wchodzi wielu niedoświadczonych pracowników. Szef kontroluje i koordynuje pracę.

Wielkość zespołu może być znacznie większa niż w strukturze sieciowej.

Duże problemy w momencie odejścia szefa zespołu.

128. W jaki sposób powinno zarządzać się modyfikacjami, jakim podlega każdy produkt programistyczny po oddaniu do eksploatacji.

Produkt oddany do eksploatacji musi podlegać zmianom. Każda modyfikacja oznacza powstanie wersji systemu, mniej lub bardziej różnej od wersji poprzedniej. Niektórzy klienci mogą nie chcieć zmiany oprogramowania, co implikuje istnienie wielu wersji produktu.

Inną przyczyną powstawania wersji jest zróżnicowanie potrzeb użytkowników. Np. mogą być wersje będące kombinacją modułów oprogramowania. Jeszcze inną przyczyną jest istnienie wielu platform sprzętowych i systemów operacyjnych.

129. **Jakiego rodzaju dokumentacja jest wytwarzana w procesie konstruowania produktu programistycznego? Jakich standardów powinna się dopracować każda firma programistyczna?**

W trakcie trwania przedsięwzięcia powstają następujące dokumenty:

* Dokumentacja procesu produkcji oprogramowania.
* Dokumentacja techniczna opisująca wytworzony produkt.

Istotne jest wypracowanie w firmie standardów dokumentacji technicznej:

* Procesów wytwarzania dokumentacji: tworzenia wstępnej wersji dokumentów, wygładzania, drukowania, powielania, oprawiania, wprowadzania zmian w istniejących dokumentach. Konieczne jest ścisłe określenie odpowiedzialnych za to osób.
* Treści i formy dokumentów: strona tytułowa, spis treści, budowa rozdziałów, podrozdziałów i sekcji, indeks, słownik.
* Sposobu dostępu do dokumentacji: niezbędne jest stworzenie rodzaju biblioteki dokumentów technicznych, z zapewnieniem sprawnego dostępu do dowolnego dokumentu.

130. **Jakim problemom musi stawić czoło zarządzanie przedsięwzięciem programistycznym?**

Celem stojącym przed kierownikiem projektu jest dostarczenie produktu w

wymaganym czasie, w ramach danego budżetu i posiadającego odpowiednią jakość.

131. **Jakie dane wejściowe są potrzebne do planowania projektu? Krótko je omów.**

* Dokument wymagań użytkownika, dokument wymagań na oprogramowanie, dokument projektu architektury systemu (zgodnie z fazą projektowania)
* Standardy w zakresie oprogramowania dla produktów i procesów wytwarzania
* Dane historyczne dla oszacowania zasobów i czasów trwania
* Dane odnośnie kosztów związanych z dostawami zewnętrznymi
* Dane odnośnie rozważanych czynników ryzyka
* Dane odnośnie środowiska wykonania, takie jak opisy nowych technologii Dane odnośnie ograniczeń czasowych, np. data dostarczenia produktu
* Dane odnośnie ograniczeń zasobów, np. dostępność personelu

132. **Jakie rodzaje zagrożeń mogą wystąpić w procesie zarządzania przedsięwzięciem programistyczny? (nie jestem pewien czy o to tutaj chodzi)**

* Czynniki doświadczenia
* Czynniki planowania
* Czynniki technologiczne
* Czynniki zewnętrzne

133**. Na czym polega zarządzanie ryzykiem i jaka jest w nim rola kierownika?**

**Zarządzanie ryzykiem polega na:**

• zredukowaniu prawdopodobieństwa wystąpienia okoliczności zagrożeń, • zminimalizowaniu skutków zagrożeń, które wystąpiły.

**Aktywności kierownika:**

• ciągłe śledzenie okoliczności, które mogą stać się zagrożeniami projektu,

• poprawianie planu celem zminimalizowania prawdopodobieństwa zagrożenia, • określenie planu awaryjnego na wypadek okoliczności zagrożenia,

• wdrożenie planu w wypadku wystąpienia okoliczności zagrażającej.

*Zarządzanie ryzykiem nigdy nie powinno zaczynać się od optymistycznego założenia „wszystko pójdzie dobrze” („jakoś to będzie”), ale raczej od pytania „co najprawdopodobniej może pójść źle?”. Nie jest to pesymizm, ale realizm.*

134. **Czy łatwo jest zmierzyć jakość oprogramowania i jaki czynniki stanowią przeszkodę w ocenie jakości?**

Istotne jest określenie kryteriów jakości i ich priorytetu. Kryteria te powinny być zawarte w dokumencie zwanym planem jakości.

Kryteria jakości:

• Zgodność z wymaganiami użytkownika • Efektywność

• Łatwość konserwacji

• Ergonomiczność

**135. W jaki sposób zarządzający przedsięwzięciem programistycznym powinni stawić czoła możliwym zagrożeniom?**

Dla zmniejszenia ryzyka personel ZJO powinien być zaangażowany w projekt programistyczny jak najwcześniej.

Powinien on sprawdzać wymagania użytkownika, plany, procedury i dokumenty na zgodność ze standardami i przyjętymi procedurami postępowania.

**136. Jak można klasyfikować potencjalnych wykonawców oprogramowania na podstawie oceny procesu wytwórczego?**

**137. Wymień środki (techniki) ułatwiające nadzór nad jakością oprogramowania.**

**138. Omów jedną z przedstawionych na wykładzie grup czynników ryzyka, opierając się na przykładach z własnego doświadczenia.**

Czynniki doświadczenia

• brak doświadczenia i/lub kwalifikacji kierownika projektu (niedoświadczony kierownik jest poważnym zagrożeniem dla projektu),

• brak doświadczenia i/lub kwalifikacji personelu (personel powinien być sprawdzony pod względem kwalifikacji, powinien być przypisany do odpowiednich zadań, ...)

• niedojrzałość dostawców (brak sukcesów w rozwijaniu podobnych projektów, brak standardów, brak certyfikatu ISO 9000, ...).

Czynniki planowania

• niedokładność metod szacowania czasu, kosztów, zasobów,

• zbyt krótka skala czasowa (niemożliwość zrównoleglenia pewnych prac),

• zbyt długa skala czasowa (zmiany wymagań, personelu, technologii),

• zależność od awarii losowych, wandalizmu i sabotażu (zniszczenie sprzętu, zniszczenie danych, itd.),

• zła lokalizacja personelu (utrudnienia w komunikacji),

• zła definicja odpowiedzialności (brak odpowiedzialnych za kluczowe zadania, wykonywanie niepotrzebnych lub drugorzędnych zadań, ...),

• częste zmiany personelu (nowy personel wymaga czasu dla zapoznania się z dotychczasowymi pracami).

Czynniki technologiczne

• nowość technologiczna (brak doświadczeń, konieczność dodatkowego wysiłku na

rozpoznanie, ...),

• niedojrzałość lub nieodpowiedniość stosowanych metod (nowe metody są często niesprawdzone, konieczne jest praktyczne doświadczenie, ...),

• niedojrzałość lub nieodpowiedniość narzędzi (personel powinien umieć je używać, mogą być nieodpowiednie w stosunku do metod, są zmieniane w trakcie projektu, ...),

• niska jakość użytego komercyjnego oprogramowania (może być przereklamowane, może nie być niezawodne, pielęgnowalne, bezpieczne, stabilne, ...),

Czynniki zewnętrzne

• niska jakość lub niestabilność wymagań użytkownika,

• słabo zdefiniowane, niestabilne lub niestandardowe interfejsy zewnętrzne,

• niska jakość lub słaba dostępność systemów zewnętrznych (od których zależy powodzenie projektu; może być konieczne rozwijanie możliwości symulujących systemy zewnętrzne).

139. **Jaką rolę w przedsięwzięciu programistycznym odgrywa bieżące raportowanie? Czego mogą dotyczyć takie raporty i jakie są metody ich przedstawiania?**

Dokładne i aktualne raportowanie jest istotne dla sterowania projektem.

Raporty odnośnie postępu prac (rutynowe, np. co miesiąc):

• stan techniczny

• stan zasobów

• stan harmonogramu • napotkane problemy • stan finansowy

Raporty odnośnie zakończonych prac

Kierownik projektu powinien potwierdzić ten raport poprzez wydanie „certyfikatu zakończonej pracy

Raporty obciążenia czasowego pracowników

Raport opisuje (z dokładnością do dni i godzin) czas przeznaczony przez poszczególnych pracowników przeznaczony lub zużyty na wykonanie poszczególnych prac.

przykłady motor raportowania: tabela postępu prac, wykres postępu prac

**140. Omów cel zarządzania konfiguracją oprogramowania.**

Celem zarządzania konfiguracją oprogramowania jest planowanie, organizowanie, sterowanie i koordynowanie działań mających na celu identyfikację, przechowywanie i zmiany oprogramowania w trakcie jego rozwoju, integracji i przekazania do użycia.

Każdy projekt musi podlegać konfiguracji oprogramowania. Ma ono krytyczny wpływ na jakość końcowego produktu. Jest niezbędne dla efektywnego rozwoju oprogramowania i jego późniejszej pielęgnacyjności.

**141. Jakim problemom musi stawiać czoło zarządzanie konfiguracją oprogramowania?**

ZKO jest szczególnie ważne, jeżeli projekt może toczyć się przez wiele lat, jeżeli cel lub wymagania na oprogramowanie są niestabilne, jeżeli oprogramowanie może mieć wielu użytkowników, i/lub jeżeli oprogramowanie jest przewidziane na wiele platform sprzętowo-programowych.

W takich sytuacjach złe zarządzanie konfiguracją oprogramowania może całkowicie sparaliżować projekt.

**142. Co jest przedmiotem zarządzania konfiguracją oprogramowania? Krótko omów te elementy.**

Wszystkie elementy projektu i oprogramowania muszą być przedmiotem ZKO, w szczególności:

* dokumentacja: wymagań, analityczna, projektowa, testowania, użytkownika, itd.
* moduły z kodem źródłowym, kody do konsolidowania, kody binarne,
* ekrany interfejsu użytkownika,
* pliki z danymi tekstowymi (np. komunikatami systemu), bazy danych, słowniki, itd.
* kompilatory, konsolidatory, interpretery, biblioteki, protokoły, narzędzia CASE, konfiguracje sprzętowe, itd.
* oprogramowanie testujące, dane testujące,
* serwery WWW wraz z odpowiednimi stronami HTML i oprogramowaniem

**143. Omów aktywności związane z zarządzaniem konfiguracją oprogramowania?**

* Identyfikacja pozycji konfiguracji (PK) Przechowywanie pozycji konfiguracji (PK)
* Kontrola zmian konfiguracji
* Określanie statusu konfiguracji
* Przekazanie pozycji konfiguracji na zewnątrz (release)

144. **Wyjaśnij pojęcie produktu bazowego w zarządzaniu konfiguracją oprogramowania**.

Produktem bazowym jest pozycja konfiguracji oceniona i zaakceptowana formalnie przez odpowiednie ciało weryfikacyjne jako zakończona, stanowiąca podstawę do dalszych faz rozwoju projektu.

**145. Wyjaśnij pojęcie wersji w zarządzaniu konfiguracją oprogramowania.**

Termin wersja (lub wariant) jest używany dla określenia pozycji konfiguracji, która ma prawie identyczną logikę i przeznaczenie, ale różni się w pewnych aspektach, takich jak:

* docelowa platforma/konfiguracja sprzętowa lub system operacyjny;
* protokół komunikacyjny, współdziałanie z innym (zewnętrznym) oprogramowaniem;
* język użytkownika (komend, komunikatów, menu), np. polski, angielski, itd.; specyficzne wymagania poszczególnych użytkowników;
* postępujące w czasie ulepszenia;
* realizacja celów diagnostycznych i testowych podczas rozwoju oprogramowania

**146. Wyjaśnij pojęcie wydania (opublikowania), związane z pozycją konfiguracji w zarządzaniu konfiguracją oprogramowania.**

Każda pozycja konfiguracji (zwykle cały projekt, ale niekoniecznie), która jest zakończona i oficjalnie przekazana na zewnątrz (zwykle na zewnątrz firmy wytwórcy oprogramowania), jest określana jako wydanie (release).

**147. W jakim celu tworzy się konwencję identyfikacji konfiguracji i co ona powinna zawierać?**

Konwencja powinna odzwierciedlać hierarchiczną strukturę pozycji konfiguracji, rodzaj pozycji i/lub przypisanie pozycji do projektu. Konwencja powinna odzwierciedlać przyjęte w danej firmie formularze, dokumenty i kody.

Konwencja identyfikacji powinna:

• ustalać jak należy nazywać pozycje konfiguracji;

• określać kto jest odpowiedzialny za nazwanie danej pozycji konfiguracji; • odwzorowywać (w miarę możności) historię danej pozycji konfiguracji.

**148. Kto i na jakim poziomie odpowiada za pozycję konfigurację?**

Trzy poziomy odpowiedzialności:

• autor kodu (programista) lub dokumentacji;

• kierownik projektu;

• ciało kontrolno-rewizyjne.

Kierownik projektu jest odpowiedzialny za połączenie PK niższego poziomu (kod, dokumentacja) w PK wyższego poziomu (konfiguracje).

**149. Omów cechy jakimi powinien charakteryzować się system przechowywania pozycji konfiguracji.**

System przechowywania PK musi dotyczyć wszystkich mediów - elektronicznych, papierowych i innych

System powinien także rejestrować i przechowywać wszelkie dokumenty administracyjne związane z projektami oprogramowania, takie jak raporty etapowe i końcowe, zlecenia, raporty zaistniałych problemów, raporty z testów, itd.

Dokumenty administracyjne powinny być powiązane z pozycjami konfiguracji w taki sposób, aby można było prześledzić ich historię oraz związki przyczynowo- skutkowe pomiędzy dokumentami i pozycjami konfiguracji.

Kluczową cechą biblioteki jest bezpieczeństwo i autoryzowany dostęp