**1.Analiza systemów informatycznych, dokument wizji, potrzeby i wymagania użytkownika.**

**2.Proces testowania oprogramowania, rodzaje testów.**

**3.Rational Unified Process**

Rational Unified Process (RUP) – proces iteracyjnego wytwarzania oprogramowania opracowany przez firmę Rational Software Corporation (przedsiębiorstwo zostało przejęte przez IBM).

Proces RUP nie jest pojedynczym, ściśle określonym procesem, ale raczej szablonem procesu. Został on zaprojektowany w celu przystosowania do charakteru konkretnej organizacji (przedsiębiorstwa), konkretnego zespołu projektowego lub nawet charakteru konkretnego projektu. Z szablonu RUP można wybrać elementy w zależności od konkretnych potrzeb.

RUP bazuje na zbiorze zasad inżynierii programowania oraz najlepszych praktykach, na przykład:

- iteracyjnym wytwarzaniu oprogramowania (Iterative Development)

- zarządzaniu wymaganiami (Requirement Management)

- używaniu architektury bazującej na komponentach (Component-based architecture)

- graficznym projektowaniu oprogramowania

- kontroli jakości oprogramowania (Quality Assurance)

- procesu kontroli zmian w oprogramowaniu (Change Management)

**4.Zwinne metody wytwarzania oprogramowania**

Grupa metod wytwarzania oprogramowania opartego na programowaniu iteracyjno-przyrostowym, powstałe jako alternatywa do tradycyjnych metod typu waterfall. Najważniejszym założeniem metodyk zwinnych jest obserwacja, że wymagania odbiorcy (klienta) często ewoluują podczas trwania projektu

Manifest Agile (ang. Agile Manifesto) – założenia:

\* osiągnięcie satysfakcji klienta poprzez szybkość wytwarzania oprogramowania,

\* działające oprogramowanie jest dostarczane okresowo (raczej tygodniowo niż miesięcznie),

\* podstawową miarą postępu jest działające oprogramowanie,

\* późne zmiany w specyfikacji nie mają destrukcyjnego wpływu na proces wytwarzania oprogramowania,

\* bliska, codzienna współpraca pomiędzy biznesem a deweloperem,

\* bezpośredni kontakt jako najlepsza forma komunikacji w zespole i poza nim,

\* ciągła uwaga nastawiona na aspekty techniczne oraz dobry projekt (design),

\* prostota,

\* samozarządzalność zespołów,

\* regularna adaptacja do zmieniających się wymagań

**5.Metoda Scrum**

**6.Zwinne estymaty projektów IT**

**7.Inteligentne klasyfikatory**

to systemy informatyczne typu: SVM, drzewa, perceptrony

**8.Maszyny z wektorami wspierającymi (SVM)**

Maszyna wektorów nośnych (maszyna wektorów podpierających), SVM (z ang. support vector machine) – abstrakcyjny koncept maszyny, która działa jak klasyfikator, a której nauka ma na celu wyznaczenie hiperpłaszczyzny rozdzielającej z maksymalnym marginesem przykłady należące do dwóch klas. Często wykorzystywana niejawnie w procesie rozpoznawania obrazów.

In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data is unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The support vector clustering[2] algorithm created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications

**9.Wybór optymalnego modelu, walidacja i walidacja krzyżowa.**

Walidacja: jak jest 3 zbiory (treningowy, testowy, walidacyjny)

Walidacja krzyżowa: m-fold CV, repeated random subsampling, leave-one-out, leave-p-out.

**10.Drzewa decyzyjne i ich nauka**

Jest to etykietowane drzewo, w którym każdy węzeł wewnętrzny odpowiada przeprowadzeniu pewnego testu na wartościach atrybutów. Z węzła wewnętrznego wychodzi tyle gałęzi, ile jest możliwych wyników testu odpowiadającego temu węzłowi, każdy liść zawiera decyzję o klasyfikacji przykładu. Typowe drzewo decyzyjne jest pokazane na rysunku 1.

Węzły drzewa decyzyjnego można też traktować jako podzbiory obiektów z tablicy. Korzeniowi odpowiada tablica oryginalna, kolejnym węzłom - te obiekty, które spełniają warunki odpowiadające gałęziom, które do nich prowadzą.

Drzewo decyzyjne buduje się w sposób rekurencyjny od korzenia do liścia, metodą "dziel i zwyciężaj". Pierwszy algorytm ID3 i jego późniejsze rozszerzenie C4.5 opracował Rossa Quinlan (profesor University of New South Wales, Australia). Ogólna idea algorytmu indukcji drzewa jest następująca:

Drzewo zaczyna od pojedynczego węzła reprezentującego cały zbiór treningowy.

Jeżeli wszystkie przykłady należą do jednej klasy decyzyjnej, to zbadany węzeł staje się liściem i jest on etykietowany tą decyzją.

W przeciwnym przypadku algorytm wykorzystuje miarę entropii (funkcja przyrostu informacji) jako heurystyki do wyboru atrybutu, który najlepiej dzieli zbiór przykładów treningowych.

Dla każdego wyniku testu tworzy się jedno odgałęzienie i przykłady treningowe są odpowiednio rozdzielone do nowych węzłów (poddrzew).

Algorytm działa dalej w rekurencyjny sposób dla zbiorów przykładów przydzielonych do poddrzew.

Algorytm kończy się, gdy kryterium stopu jest spełnione.

Niżej przedstawiono szkic algorytmu indukcji drzewa decyzyjnego z danych przykładowych.

**11.Naiwny klasyfikator bayesowski, estymacja funkcji gęstości prawdopodobieństwa.**

Naiwny klasyfikator bayesowski – prosty klasyfikator probabilistyczny. Naiwne klasyfikatory bayesowskie są oparte na założeniu o wzajemnej niezależności predyktorów (zmiennych niezależnych). Często nie mają one żadnego związku z rzeczywistością i właśnie z tego powodu nazywa się je naiwnymi. Bardziej opisowe jest określenie – „model cech niezależnych”. Ponadto model prawdopodobieństwa można wyprowadzić korzystając z twierdzenia Bayesa.

**12.Złożoność obliczeniowa: pesymistyczna, średnia, optymistyczna.**

Ilość zasobów niezbędnych do wykonania algorytmu można rozumieć jako jego złożoność. W zależności od rozważanego zasobu mówimy o złożoności czasowej czy też złożoności pamięciowej. Oczywiście w większości wypadków ilość potrzebnych zasobów będzie się różnić w zależności od danych wejściowych z zakresu danego zagadnienia.

Przykładowo można by rozpatrzyć rozkład liczb na czynniki pierwsze. Przewidzieć można, że (niezależnie od zastosowanego algorytmu) im większa liczba, tym więcej zasobów będzie potrzebnych do jej rozłożenia. Tę cechę podziela większość zagadnień obliczeniowych – im większe rozmiary danych wejściowych, tym więcej zasobów (czasu, procesorów, pamięci) jest koniecznych do wykonania danych obliczeń. Złożoność algorytmu jest więc funkcją rozmiaru danych wejściowych.

Kolejnym problemem jest fakt, iż złożoność zwykle nie zależy wyłącznie od rozmiaru danych, ale może się znacznie różnić dla danych wejściowych o identycznym rozmiarze. Dwoma często stosowanymi sposobami podejścia są:

rozpatrywanie przypadków najgorszych – złożoność pesymistyczna

zastosowanie określonego sposobu uśrednienia wszystkich możliwych przypadków – złożoność oczekiwana

Przykłady złożoności obliczeniowych:

O(1): złożonośc stała, brak zależności od rozmiaru zadania;

O(log(n)): złożoność logarytmiczna, np. wyszukiwanie binarne w tablicy uporza˛dkowanej;

O(n): złożonośc liniowa, np. iloczyn skalarny wektorów;

O(nlog(n)): złożonośc liniowo-logarytmiczna, cechuje np. algorytmy opracowane

wg strategii "dziel i rza˛dz´" z podziałem na dwa zadania o złożoności dwukrotnie

mniejszej niż zadanie wejściowe;

O(n2): złożonośc kwadratowa, np. mnożenie macierzy przez wektor;

O(n3);O(n4); :::: złożonośc wielomianowa;

O(2n): złożonośc wykładnicza;

O(n!): złożonośc wykładnicza silnia, tj. wymagane jest sprawdzenie wszystkich permutacji

elementów zbioru.

**13. Strategie konstrukcyjne algorytmów (dziel i rządź, rekurencja, programowanie dynamiczne, …).**

Dziel i rządź – podział problemu na mniejsze podproblemy.

Rekurencja – odwołanie funkcji do samej siebie

Programowanie dynamiczne – zapamiętywanie wyników pośrednich w celu usprawnienia.

**14. Programowanie masowo-równoległe (procesory GPU)**

GPU (ang. Graphics Processing Units) to główne układy kart graficznych będące wyspecjalizowanymi akceleratory sprzętowymi, które stworzono z myślą o sprzętowym przyśpieszaniu operacji graficznych: przetwarzanie geometrii, teksturowanie wielokątów,

renderowanie oświetlenia, itp..

Wspomniane operacje graficzne wymagają stosunkowo prostych obliczeń matematycznych (przekształcenia afiniczne, interpolacja), ale zastosowanych do wielu elementów jednocześnie.

Stąd układy GPU są projektowane z myślą o równoległym wykonywaniu wielu prostych operacji arytmetycznych, często z dokładnością pojedynczej precyzji float32.

Stąd narodził się pomysł wykorzystania potencjału układów GPU do wykonywania obliczeń ogólnego przeznaczenia GP (ang. General Purpose).

Pomysł ten skutkował stworzeniem biblioteki programistycznej dla obliczeń GP w roku 2007. Mowa tutaj o środowisku CUDA (ang. Compute Unified Device Architecture).

CUDA jest oparta na języku programowania C++ i stanowi środowisko programistyczne dla obliczeń masowo-równoległych GP na kartach graficznych firmy NVIDIA.

**15. Warstwowa budowa aplikacji webowych**

3 warswy: danych → przetwarzania → prezentacji

**16. Usługi sieciowe typu SOAP, RESTful**

A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards.— W3C, Web Services Glossary[3]

Usługa sieciowa (web service) to

**Web Service** (usługa webowa/sieciowa) najłatwiej  
jest zdefiniować jako niezależny od platformy  
i implementacji komponent, którego zadaniem jest  
dostarczenie konkretnej funkcjonalności. Rozwijając  
słowo \”niezależny\”, warto zauważyć, że liczba  
wspieranych platform jest naprawdę imponująca,  
a można do nich zaliczyć nie tylko Javę i .NET, ale  
także języki PHP, Python, Ruby i wiele innych.  
Ponadto dzięki otwartej specyfikacji powstało  
wiele frameworków, które w znaczny sposób  
uproszczają modularyzację aplikacji i szybkie tworzenie  
nawet złożonych usług webowych.

<https://magazynt3.pl/Uslugi-webowe-w-praktyce/>

Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) are two answers to the same question: how to access Web services. The choice initially may seem easy, but at times it can be surprisingly difficult.

SOAP is a standards-based Web services access protocol that has been around for a while and enjoys all of the benefits of long-term use. Originally developed by Microsoft, SOAP really isn’t as simple as the acronym would suggest.

REST is the newcomer to the block. It seeks to fix the problems with SOAP and provide a truly simple method of accessing Web services. However, sometimes SOAP is actually easier to use; sometimes REST has problems of its own. Both techniques have issues to consider when deciding which protocol to use.

**17. Liniowa i nieliniowa filtracja obrazu**

Filtry liniowe – takie, które spełniają warunki:

addytywna – phi(f + g) = phi(f) + phi(g)

jednorodna – lam \* phi(f) = phi(lam \* f)

Przykłady: dolnoprzepustowe, górnoprzepustowe.

Reszta nieliniowa. Przykładowo: medianowy/minimalny/maksymalny, logiczne, adaptacyjne.

**18. Atomowość instrukcji, przeplot danych**

Operacje atomowe – są to operacje o sekwencji działań odczyt-modyfikacja-zapis wykonywane w sposób niepodzielny. Operacje atomowe dotyczą działań na pamięci globalnej lub pamięci

współdzielonej.

Atomowy charakter operacji sprawia, iż żaden inny wątek nie ma dostępu do tego samego adresu w pamięci dopóki operacja atomowa nie zostanie zakończona, tzn. wynik nie zostanie zapisany pod tym adresem.

Operacje atomowe w rozumieniu ogólnym nie stanowią barier synchronizacyjnych, nie wymuszają określonego uszeregowania wątków, jak również nie stanowią barier (ang. fence) dla transakcji

dostępu do pamięci.

Przykłady: arytmetyczne, bitowe

**19. Mechanizm wirtualnej pamięci wspólnej w programowaniu równoległym.**

Pamięć współdzielona, wspólna (ang. Shared Memory) stanowi obszar pamięci fizycznie zlokalizowany w obrębie procesora SM. Ze względu na umiejscowienie dostęp do tej pamięci może być realizowany wyłącznie przez wątki należące do tego samego bloku obliczeniowego.

Nawet dziesięciokrotne przyśpieszenie dostępu

**20. Miary zrównoleglenia programów­­­**