Dinamički web sistemi - web aplikacije koje imaju dinamički sadržaj koji se razlikuje za svakog korisnika (u zavisnosti od interakcije sa korisnicima). Dobijaju personalizovanu stranicu koja je interaktivna i ima real-time ponašanje; E-bay, Netflix, Google, Instagram...

Web sistemi - skupina tehnologija, protokola i aplikacija koji omogućavaju interakciju i razmjenu informacija putem interneta. Koriste se za pružanje usluga kao web stranice, e-trgovina, e-pošta itd. Sastoje se od ključnih komponenata kao što su klijentski uređaji, web stranice, web serveri, HTTP(Hypertext Transfer protocol), baze podataka i same web aplikacije.

Značaj web sistema - web sistemi su važni u današnjem digitalnom svijetu jer omogućavaju komunikaciju, razmjenu informacija, pružanje različitih usluga i suradnju klijenata na globalnom nivou.

Arhitektura web sistema - dijeli se na tradicionalnu i modernu arhitekturu sa više detalja

Tradicionalna arhutektura web sistema - (klijent – server arhitektura) je sastavljena od servera, klijenata i baze podataka.

Moderna arhitektura sa više detalja:

- DNS (povezuje domene i IP adrese)
- Load Balancer (radi horizontalno skaliranje i više mirror kopija servera)
- Web app servers (procesuira zahtjeve i vraća response)
- Baze podataka (čuvaju i pretražuju podatke)
- Caching service (kešira podatke i omogućava tako bržu pretragu za buduće zahtjeve)
- Job queue (nije obavezan ali služi za poslove manjeg prioriteta)
- Full-text search service (radi pretragu po tekstu te vraća najviše sličnih rezultata)
- Servisi (služe kao odvojene aplikacije a koriste se za web sisteme)
- Data warehouse (čuvanje i analiziranje raznih podataka iz web sistema),
- CDN (HTML, CSS, JS i slike)

Modeli komponenti web sistema - postoje 3 modela a to su: jedan server i jedna baza podataka, više servera i jedna baza podataka te više servera i više baza podataka.

Monolitni sistemi - vrsta softverske arhitekture u kojoj je cijela aplikacija implementirana kao jedna velika nedjeljiva cjelina, a u njemu sve komponente i funkcionalnosti su integrirane i izvršavaju se unutar istog izvršnog okruženja ili procesa. Karakteristike ovog sistema su jedinsveno okruženje, sav razvoj, održavanje i razvijanje se radi kao jedna cjelina, koristi se jedan programski jezik i set tehnologija za implementaciju svih komponenti, a samim tim povećana je i složenost jer je teško izolirati i testirati pojedinačne komponente. Prednost ovakvih sistema je jednostavan razvoj i razumijevanje, te bržu inicijalnu implemenaciju.

Aplikacijski servisi - servisi koji se dijele na mikroservise i serverless aplikacije.

Mikroservisi - pristup razvoju softvera u kojem se aplikacija razbija na manje nezavisne servise koji rade zajedno kao distribuirani sistemi.

Serverless aplikacije - aplikacije u kojima se infrastruktura i upravljanje serverima prepušta cloud provajderu, te tako omogućava da se developeri fokusiraju na pisanje koda i funkcionalnosti same aplikacije a ne upravljanjem infrastrukture.

Frontend arhitektura - dijeli se na Single-page aplikaciju(SPA) i Server-Side rendered aplikacije(SSR).

Single-page aplikacije - vrsta web aplikacija koje se učitavaju jednom i sva interakcija sa korisnikom se odvija bez ponovnog učitavanja stranica.

Server-side rendered aplikacije - web aplikacije u kojima se generisanje prvog HTML sadržaja odvija na strani servera, te se nakon toga koristi dinamičko ažuriranje stranice kako bi se poboljšao korisničko iskustvo.

Backend arhitektura - mikroservisi i serverless.

Mikroservisi - servisi koji rješavaju jednu funkcionalnost te imaju brži razvoj od drugih arhitektura. Obično se za njihovu implementaciju koriste različite platforme i jezici a predstavljaju jako efikasnu i produktivnu metodu.

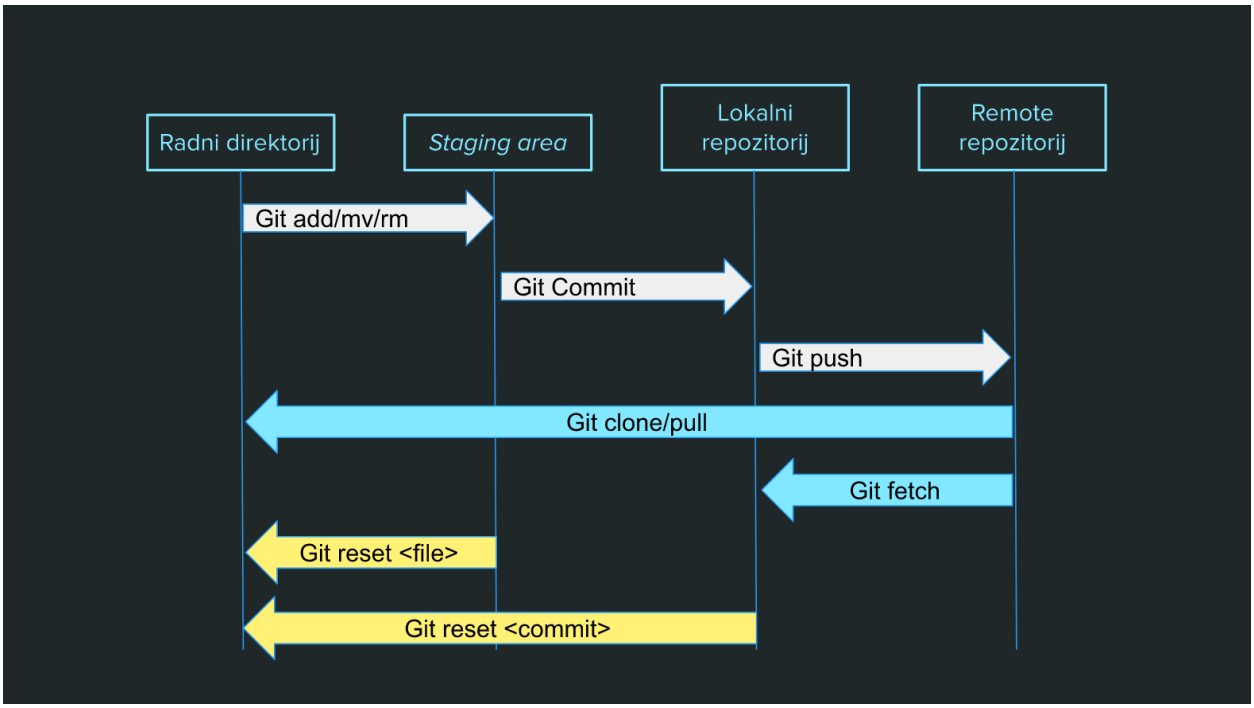
Serverless - Za ove servise nije nephodna konfiguracija i administracija a serveri i infrastrukture se iznajmljuju. Obično je riječ o AWS ili Google Cloudu.

Baze podataka - imaju svoje servere baza podataka i služe za održavanje sistema sa svim podacima koje aplikacija treba pratiti. Jako je bitno raditi njihov backup pošto se u bazama podataka kriju sve informacije koje ne smijemo izgubiti.

Osobine dobre arhitekture - efikasna i fleksibilna u razvoju, brzo i efikasno se testira, razumljiv kod za čitanje, brzo vrijeme odgovora, skalabilno i stabilno, jednostavno, sigurno i za developera i za klijenta.

VSC (Version Control System) - sistem koji omogućava verzioniranje koda, a to znači proces praćenja i upravljanja promjena u kodu kroz vrijeme, efikasnu saradnju i održavanje koda te minimiziranje rizika od greške ili gubitka podataka. Sam značaj ovoga jeste praćenje promjena, rješavanje konflikta između dva različita koda i olakšana saradnja, omogućeni povratak na prethodne verzije i rad velikog broja programera na istom kodu.

GIT - sistem/alat za upravljanje promjenama u kodu. Prvobitno razvijan za potrebe Linuxa, a sada je dostupan i za Windows, Linux i MacOS. Ima open-source kod a jako je pouzdan, popularan i što je najvažnije brz. Za ciljeve ima omogućavanje lakšeg razvoja kompleksnog softverskog sistema ili softvera pri radu više developera kao i efikasno upravljanje različitim verzijama koda. Poznati git serveri su GitHub, BitBucket i GitLab. U njemu postoji repozitorij koji predstavlja skup fajlova koji su u sistemu verzioniranja i čine jedan projekat. Repozitoriji se dalje dijele na globalne (koji su na git serveru) i lokalne (koji su kopija globalnog i svaki developer sadrži kopiju lokalno na svom računaru). Postoje neke 4 osnovne podjele koje se nalaze na slici:



Grane/Branching - projekat sadrži glavnu granu koja se naziva master ili main. Ona ima projekat koji je funkcionalan. Developeri nove funkcionalnosti implementiraju na novim granama, koje u početku sadrže kopiju master grane ali je ta grana potpune neovisna od same master grane. Promjene i modifikacije koda rade se commitom i pushom svog novog koda na glavnu granu. Toj grani koji developer kreira mogu se pridružiti i ostali u timu. Kada je funkcionalnost u potpunosti gotova, developer obavijesti druge u timu odnosto napravi pull request, te nakon revizije koda, obično iskusnijeg programera, ta nova grana se integrira u glavnu i nestaje. Dakle, git je distriburiani sistem za upravljanje izvornim kodom u kojem svaki korisnik ima svoju kopiju cijele historije izmjena koja olakšava suradnju između developera a omogućava i rad bez interneske veze.

GIT Copilot - AI alat za asistenciju developerima koji kreira prijedloge koda i završava kod umjesto developera.

Softverski inžinjering - disciplina koja se bavi primjenom inžinjerskih principa i metodologija na razvoj softvera a obuhvata cjelokupni proces razvoja softvera.

Proces razvoja softvera - proces razvoja sofvtera sastavljen je od više faza a to su analiza zahtjeva, dizajn, implementacija, testiranje, održavanje i nadogradnja (ovim redom).

Modeli razvoja softvera - waterfall metoda, agilne medote i scrum metode.

Tipovi softverskih timova - horizontalni timovi (timovi se dijele na timove za frontend, bazu, backend, ai...), vertikalni timovi (timovi se dijele na timove za razvoj, timove za održavanje...) te hibridni timovi koji su kombinacija horizontalnih i vertikalnih.

Strukture softverskih timova - hijerarhijska struktura (u timu postoji menadžer projekta, vođa tima, obični programer...), ravnopravna struktura (scrum timovi) i matrica struktura (struktura u kojoj programer radi različite projekte istovremeno).

Alati - alati za menadžment su obično Jira i Trello, za komunikaciju Slack i Teams a za suradnju Github, Gitlab.

Model vodopada - waterfall model radi na principu da se na sljedeći segment razvoja aplikacije prelazi tek kada je jedan gotov u potpunosti. Na primjer prvo skupimo sve potrebe aplikacije, onda u ptpunosti dizajniramo aplikaciju, pa u postpunosti završimo izgradnju aplikacije, pa testiramo itd... Ovaj metod je poprilično zastario obzirom da je jako spor, ali je idalje uredu koristiti ga za manje projekte ili za privatne projekte.

Agilne metode - omogućavaju brzo prilagođavanje promjenama u zahtjevima i imaju fleksibilan pristup razvoju softvera, a koriste se kada su zahtjevi i ciljevi projekta nedefinisani u potpunosti, što znači u moderno vrijeme uvijek. Postoji agilni manifest koji je skup od 12 principa definisanih 2001. godine, a cilj je razvijati funkcionalan softvera uz veliku mogućnost prilagođavanja.

Agilne metode u praksi - scrum, kanban i extreme programming.

Scrum - agilna metodologija upravljanja projektima koja se temelji na timskom iterativnom razvoju aplikacija uz kontinualno poboljšanje. Sastoji se od definisanja uloga, ceremonija i artefakata, a cilj je transparentnost koda, brza isporuka koda i prilagodljivost u promjenjivim okruženjima.

Sprint - sprint je vremenski okvir scrum metodologije, a definira period u kojem se tim fokusira na određeni skup poslova i ciljeva. Obično je vrijeme izmedju 1 do 4 sedmice, a najčešće 2 sedmice. Tokom sprinta tim radi na implementaciji odredjenih zadataka koji se definiraju na početku svakog sprinta. Tokom sprinta timovi održavaju sastanke da bi se dijelile informacije o napretku i prepoznali prepreke. Postoji i sprint retrospektiva koja se radi kako bi se analizirao rad i identificirali načini za poboljšavanje u budućnosti.

Product Backlog - ceremonija scrum metodologije. Predstavlja listu svih zadataka, funkcionalnosti i zahtjeva koji su potrebni za razvoj proizvoda. On se kontinuirano nadopunjuje i ažurira tijekom rada ciklusa.

Sprint Backlog - lista zadataka koji se moraju uraditi tokom nekog sprinta, a kreira se na početku svakog sprinta obično na sastanku tima, gdje tim zajedno odabere zadatke i procjenjuje vrijeme potrebno za izvršavanje.

Scrum master - uloga u scrum metodologiji koja je odgovorna za osiguranje primjene scrum principa i procesa u timu. Ima zadatak podržavanja tima u postizanju učinkovitosti i uspješnu primjenu scrum metodologije. Uloga im je vođenje svih ceremonija kao daily scrum, sprint rewiev itd.

Daily Scrum - svakodnevni sprint sastanak koji traje do 15ak minuta. Svaki član tima odgovara na 3 pitanja. To su šta sam radio, šta ću raditi i šta me muči. Cilj ovih sastanaka je da se tim sinhronizira i otkriju se mogući probleni i poboljšava se suradnja.

Sprint planning - ceremonija scrum metodologije u kojoj tim i product owner zajedno definišu ciljeve i biraju zadatke za naredni sprint, a obično se održavaju na početku svakog sprinta. Tu product owner objašnjava prioritete, ciljeve i funkcionalnosti koje želi da se implementiraju.

Sprint review - ceremonija scrum metodologije koja se održava na kraju svakog sprinta kako bi se pregledao rad i prikazali rezultati koji su relevantni za product ownera, koji učestvuje u sprint reviewu i daje povratne informacije ali i diskutuje s timom. Predstavlja priliku tima da demonstrira funkcionalnosti i ostvarene ciljeve. Cilj je provjeriti da li je ostvareno ono što je planirano.

Sprint retrospective - ceremonija scrum metodologije koja se država nakon sprinta a u njima se tim reflektuje na svoj rad, razgovara o preprekama na koje su nailazili i kako su se nosili sa njima.

Definition of done (DoD) - kriterij koji se koristi u scrum metodologiji kako bi se definiralo kada se smatra da je zadatak ili funkcionalnost spreman za isporuku. DoD je dogovor između product ownera i tima o minimalnim standardima kvaliteta koje se trebaju postići, a obuhvata funkcionalne zahtjeve, testiranje, dizajn, dokumentaciju i druge specifične potrebe. Svaki zadatak ili funkcionalnost mora zadovoljiti sve DoD kriterije kako bi se smatrao završenim.

Lean metodologija - pristup organizacije koja se temelji na načinu na koji je Toyota radila 50ih godina. Glavni cilj ove metodologije jeste eliminisati beskorisne aktivnosti odnosno gubitke i optimizirati procese zarad veće učinkovitosti, kvaliteta i zadovoljstva krajnjih korisnika. Glavni koncepti ove metodologije su vrijednost (određuje se vrijednost proizvoda iz ugla kupca), protok vrijednosti (procjena vrijednosti svakog koraka proizvodnje), povlačenje (samo kada je neophodno, nema proizvodnje unaprijed), kontirano poboljšanje (uklanjanje otpada).

Kanban - agilna metoda koja se koristi za upravljanje radnim zadacima i vizualizaciju toka rada. Osnovna ideja kanbana je da se zadaci vizualno prikazuju na ploči (kanban ploča) koja ima kolone koje predstavljaju različite faze zadataka (recimo to do, in progress, done). To omogućava kontinuirani napredak, međutim potrebno je uvesti ograničenja za work in progress (wip) kako bi se ograničio broj istovremeno pokrenutih zadataka u cilju održavanja ravnoteže rada. Kanban metodologija potiče kontinuiranu optimizaciju, smanjenje preopterećenosti i fokus na stalno poboljšanje. Na kanban ploči se obično nalaze 4 kolone, a to su: To-do list, work-in-progress, validate i complete. Fokus ovakve metodologije je završavanje zadatka koji je nekom dodijeljen, a pogodan je za projekte sa potrebom za stalnim rezultatima.

Razlike kanbana i scruma:

	Scrum	Kanban
Origin	Software development	Lean manufacturing
Ideology	Learn through experiences, self-organize and prioritize, and reflect on wins and losses to continuously improve.	Use visuals to improve work-in-progress
Cadence	Regular, fixed-length sprints (i.e. two weeks)	Continuous flow
Practice	Sprint planning, sprint, daily scrum, sprint review, sprint retrospective	Visualize the flow of work, limit work-in-progress, manage flow, incorporate feedback loops
Roles	Product owner, scrum master, development team	No required roles

XP (Extreme programming) - cilj ove metodologije je brza isporuka kvalitetnog softvera a radi se u projektima sa brzim promjenama zahtjeva. Tačnije, obično se radi u projektima koji su istraživačkog tipa jer nikada se ne zna kada će se otkriti nešto novo, odnosno tok samog projekta je jako neizvjesan. Prednosti ove metodologije su poboljšanje kvaliteta, veća produktivnost, bolja komunikacija i manji rizik bugova. Kako se te stvari postižu? U suštini pair-programmingom za produkcijski kod, odnosno 2 osobe programiraju uporedo. Na taj način imaju komunikaciju te brže uklanjaju bugove. Također, ukoliko je jedna osoba odsutna, druga zna odakle treba nastaviti i ne čeka cijeli projekat na tu osobu. Prije produkcijskog koda rade se unit testovi. Unit testovi su testovi koji odvojeno testiraju dio svakog koda kako bi se provjerila njegova ispravnost, a programeri pišu automatske testove koji provjeravaju očekivane rezultate za određeni dio koda. Kontinualna integracija je praksa u kojoj se integriraju promjene u zajednički repozitorij a osigurava se da promjene budu međusobno spojene bez konflikta. Refaktorisanje koda je još jedna praksa XP-a koja se odnosi na restrukturiranje koda bez mijenjanja njegove funkcionalnosti a radi se radi poboljšanja čitljivosti koda i samog kvaliteta istog. Pored ovih koncepata, bitno je naglasiti da je ovdje odgovornost koda, a samim tim i vlasništvo koda grupno, odnosno od osoba koje zajedno rade u timu.

O metodologijama - potrebno je da vođa projekta ima fleksibilan pristup razvoju te da može metodologiju primijeniti korektno i timu i tipu samog projekta. Da se stvari koje radi i traži ne primjenjuju izolirano. Svaka od 3 navedene agilne metode ima i prednosti i mane, a njihove kombinacije dovode do uspjeha.