

Dinamički web sistemi – Web aplikacije koje imaju dinamički sadržaj koji se razlikuje u zavisnosti od interakcije sa korisnicima te dobijaju personalizovanu stranicu koja je interaktivna I ima real-time ponašanje, tipa ebay, netflix, google, instagram...

Web sistemi – skupin tehnologija, protokola I aplikacija koji omogućavaju interakciju I razmijenu 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 uređaju, 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, razmijenu informacija, pružanje različitih usluga I suradnju klijenata na globalnom nivou.

Arhitektura web sistem – Dijeli se na tradicionalnu, modernom arhitekturom sa više detalja

Tradicionalna arhitektura web sistema – To jeste klijent – server arhitektura koja 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(Procesira 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 jedna baza podataka, više servera I jedna baza podataka I više servera 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 ade 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 koje 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.

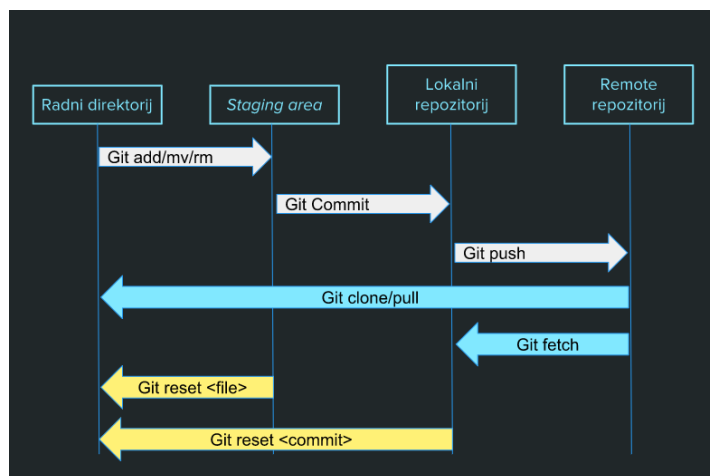
Serverless – Za ove servise nije neophodna konfiguracija i administracija a serveri i infrastrukture se iznajmljuju. Obično je riječ o AWS ili Google Cloudu

Baze podataka – Imaju svoje servere baze podataka i služe za održavanje sistema sa svim podacima koje aplikacija treba pratiti. Jako je bitno raditi njihov backup posto 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 promijenama u kodu kroz vrijeme, efikasnu saradnju i održavanje koda i minimiziranje rizika od greške i gubitka podataka. Sam značaj ovoga jeste praćenje promjena, rješavanje konflikata 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 promijenama u kodu. Prvobitno razvijen za potrebe linkusa, 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 potpuno neovisna od same master grane. Promjene i modifikacije koda rade se commitom i pushom svoga kopog 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 odnosno napravi pull request, te nakon revizije koda, obično iskusnijeg programera, ta nova grana se integrira u glavnu i nestaje.

Dakle, git je distribuirani sistem za upravljanje izvornim kodu kojem svaki korisnik ima svoju kopiju cijele historije izmjena koja olakšava lakšu suradnju između developera a omogućava i rad bez interne veze.

GIT Coplito – AI alat za asistenciju developerima koji kreira predloge koda i završava kod umjesto developera.

Šta je softverski inženjering - Disciplina koja se bavi primjenom inženjerskih principa i metodologija na razvoj softvera a obuhvata cjelokupni proces razvoja softvera.

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

Modeli razvoja softvera – Waterfall metoda, agilne metode 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, Teams a 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 potpunosti dizajniramo aplikaciju, pa u potpunosti završimo izgradnju aplikacije, pa testiramo itd... Ovaj metod je poprilično zastario obzirom da je jako spor, ali je i dalje 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 nedefinirani u potpunosti, što znači u moderno vrijeme uvijek. Postoji neki agilni manifest koji je skup od 12 principa definisanih 2001. Godine, a cilj je razvijati funkcionalan softver 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 definiranja 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 između 1 do 4 sedmice, a najčešće 2 sedmice. Tokom sprinta tim radi na implementaciji određenih 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 review itd.

Daily Scrum – svakodnevni sprint sastanak koji traje do 15 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 problemi 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 i se pregledao rad i prikazali rezultati koji su relevantni za product ownera, koji učestvuje u sprint reviewu i daje povratne informacije, kao i što 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 održava nakon sprinta a u njima se tim reflektuje na svoj rad, razgovara o preprekama na koje su naišli 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 50thi 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 vrijednost svakog koraka proizvodnje), povlačenje(samo kada je neophodno, nema proizvodnje unaprijed), kontinuirano 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 work in progress(wip) kako bi se ograničio broj istovremeno pokrenutih zadataka kako bi se održala ravnotežau radu. 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 dodjeljen, a pogodan je za projekte sa potrebom za stalnim rezultatima.

Razlike kanban I scrum:

	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 na 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, obržavanju I samom kvalitetu 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.