PITANJA ZA VISE PROGRAMSKE JEZIKE I RAD ALATE

1. Koje predviđanje je formulisano „Murovim zakonom“	Gordon Mur, jedan od osnivača Intela, predvideo je 1965. da će se broj tranzistora na čipu udvostručavati približno svake dve godine.
2. Kako možemo formulisati problem kompleksnosti softvera u odnosu na ljudske sposobnosti?	Iako kompleksnost softverskih sistema raste, ljudska sposobnost da se nosi sa tim sistemima ostaje ograničena. To važi za dizajnere i krajnje korisnike, zbog čega razvoj i korišćenje kompleksnih softvera postaju skloniji greškama.
3. Koja je osnovna namena modela razvoja softvera?	Model razvoja softvera postavlja redosled faza i kriterijume za prelazak između njih. To uključuje utvrđivanje kada je trenutna faza završena i kada treba započeti sledeću fazu. Ukratko, model odgovara na pitanja: "Do kada traje trenutna aktivnost?" i "Šta je sledeće?"
4. Koliko koraka u razvoju softvera se predviđa metodom „implementiraj i popravi“?	Metoda „implementiraj i popravi“ ima dva koraka: 1. Napiši kod. 2. Identifikuj i reši probleme u kodu. Prvo se vrši implementacija, a tek potom se razmatraju pitanja specifikacije, dizajna, testiranja i održavanja.
5. Osnovni nedostaci modela „implementiraj i popravi“?	• Posle mnogih izmena, kod postaje loše strukturiran, što otežava dalje promene. • Čak i ako je dizajn dobar, nema garancije da će softver ispuniti zahteve korisnika. • Izmene su otežane jer kod nije prilagođen za testiranje i prilagođavanja.
6. Faze u sekvencijalnom modelu razvoja softvera:	• Studija izvodljivosti • Specifikacija zahteva • Dizajn • Implementacija • Testiranje
7. Koje su aktivnosti sastavni deo faze zahteva u sekvencijalnom modelu razvoja softvera?	Faza zahteva uključuje dve aktivnosti: a) Prikupljanje i analiza zahteva b) Specifikacija zahteva

8. Integraciono testiranje u okviru sekvencijalnog modela razvoja softvera:	Integraciono testiranje podrazumeva kombinovanje modula i proveru njihove ispravnosti u zajedničkom radu. Programer integriše module jedan po jedan, što olakšava lociranje i ispravljanje grešaka. Na kraju, ceo sistem se testira uz pomoć uzoraka podataka.
9. Testiranje sistema u okviru sekvencijalnog modela razvoja softvera:	Testiranje sistema osigurava da sistem ispunjava zahteve navedene u SRS dokumentu. Ova faza se sprovodi nakon integracije svih modula. Na kraju testiranja, sistem se isporučuje kupcu.
10. Šta je primarni nedostatak sekvencijalnog modela razvoja softvera?	Sekvencijalni model zahteva potpuno razrađenu dokumentaciju kao kriterijum za završavanje faza specifikacije i dizajna, što je poznato kao „document-driven“ pristup. Ovaj zahtev je odgovarajući za softver kao što su kompajleri i sigurnosni sistemi, gde je moguće detaljno definisati zahteve unapred. Međutim, za mnoge druge vrste softvera, kao što su korisnički interfejsi, zahtev za razrađenom specifikacijom može biti neadekvatan i može dovesti do dizajniranja i pisanja neupotrebljivog koda.
11. Šta uključuje svaka iteracija u okviru iterativnog modela razvoja softvera?	Svaka iteracija obuhvata sve faze razvoja softvera: • Prikupljanje i analiza zahteva • Priprema specifikacije • Implementacija • Testiranje i pokretanje
12. Šta predstavlja „herojski“ pristup u razvoju softvera?	„Herojski“ pristup u razvoju softvera naglašava da je razvojni proces važan, ali ne presudan za uspeh projekata. Problemi u razvoju nisu uvek jasno definisani i ne mogu se uvek formalizovati. Razvoj softvera je adaptivan proces u kojem je najvažniji faktor čovek. „Heroj“ je stručnjak koji preuzima inicijativu u rešavanju nestandardnih problema i spreman je da izađe iz okvira formalizovanih modela. „Herojstvo“ može biti racionalno, ali i patološko.
NISTA	

13. **Koja je razlika između pristupa „Kontrolisanja procesa“ i „Kaubojskog“ pristupa u razvoju softvera?**	**Kontrolisanje procesa**: • Pretpostavka: Kvalitet softverskog proizvoda može se postići kontrolom kvaliteta procesa razvoja. • Fokus je na procesu, a ne na pojedincima („herojima“). • Modelovanje procesa je ključno za uspeh projekta. • Glavna uloga pripada menadžerima na vrhu hijerarhije. **Kaubojskog pristupa**: • Fokusira se na fleksibilnost i improvizaciju, često zanemarujući formalizovane procese. • Pojedinci (heroji) preuzimaju inicijativu u rešavanju problema. • Proces nije strogo definisan; rad se često prilagođava trenutnim potrebama i okolnostima.
14. Šta je prototip i čemu služi u procesu razvoja softvera?	Prototip je radni model softverskog proizvoda koji služi za: • Prikupljanje povratnih informacija • Usavršavanje dizajna pre nego što se razvije finalni proizvod Model izrade prototipa je posebno koristan kada: • Zahtevi nisu dobro definisani • Zahtevi će se verovatno menjati tokom razvoja Prototip može pomoći u smanjenju rizika da softver ne zadovolji potrebe korisnika ili zainteresovanih strana, kao i u smanjenju vremena i troškova razvoja.
15. Koji se pristupi mogu koristiti prilikom razvoja prototipa?	Prototip se može razviti koristeći različite pristupe, kao što su: • Odbačivi prototip • Evolutivni prototip • Inkrementalni prototip
16. Šta su osnovne karakteristike odbacivog prototipa	Odbaciv prototip je brz radni model koji se kreira kako bi se vizuelno prikazao izgled sistema prema korisničkim zahtevima. Ovaj model se koristi za preispitivanje i razjašnjavanje zahteva korisnika, a nakon što postigne svoju svrhu, odbacuje se, dok se sistem formalno razvija na osnovu identifikovanih zahteva.

17. Koje su prednosti prototipskog modela razvoja softvera?	Prototipski model omogućava kupcima da vide delimičan proizvod u ranoj fazi, što povećava njihovo zadovoljstvo i udobnost. Novi zahtevi se lako mogu uvažiti zahvaljujući mogućnosti usavršavanja modela, a funkcionalnosti koje nedostaju brzo se otkrivaju. Greške se identifikuju ranije, što štedi vreme i troškove, te poboljšava kvalitet softvera. Razvijeni prototip može se ponovo koristiti za složenije projekte u budućnosti. Fleksibilnost u dizajnu i rane povratne informacije od kupaca pomažu u vođenju procesa razvoja, osiguravajući da konačni proizvod ispunjava njihove potrebe. Izrada prototipa omogućava testiranje i validaciju dizajnerskih odluka, smanjuje rizik od neuspjeha projekta i poboljšava komunikaciju i saradnju među članovima tima.
18. Faze razvoja softvera primenom RAD metodologije:	• Prikupljanje zahteva • Analiza i planiranje • Projektovanje • Izgradnja i implementacija
19. Koje pretpostavke vezane za projekat razvoja softvera čine opravdanim implementaciju RAD metodologije?	Implementacija RAD metodologije je opravdana kada su zahtevi potpuno razumljivi i kada je pristup konstrukciji zasnovan na komponentama usvojen. Takođe, uslov za primenu ovog modela je da se projekat može podeliti na male module, pri čemu svaki modul može biti dodeljen posebnom razvojnom timu.
20. **Šta su to RAD alati?**	RAD alati (Rapid Application Development) su softverske razvojne platforme i radni okviri koji su dizajnirani da ubrzaju proces razvoja aplikacija.
21. Neke prednosti primjene RAD alata u odnosu na tradicionalne alate su:	**Vizuelni (grafički) interfejs**: Omogućava korisnicima da projektuju i razvijaju aplikacije bez pisanja koda, što je posebno korisno za korisnike koji nisu iskusni programeri. **Prevuci-i-pusti (drag-and-drop) komponente**: Ove komponente omogućavaju brzo i jednostavno sastavljanje aplikacija, štedeći značajno vreme i trud programerima, te pomažu u smanjenju grešaka.

22. Koju mogućnost pružaju „Low-Code“ razvojne platforme?	„Low-Code“ razvojne platforme omogućavaju programerima da kreiraju aplikacije uz minimalno manuelno kodiranje. Da bi pojednostavile razvoj aplikacija, one pružaju vizuelne (grafičke) interfejse, unapred izgrađene komponente i skript jezike visokog nivoa. Ove platforme su idealne za preduzeća koja žele brzo da izgrade funkcionalne aplikacije uz mali razvojni trud.
23. **Koju mogućnost pružaju „No-Code“ razvojne platforme?**	„No-Code“ razvojne platforme omogućavaju kreiranje aplikacija bez manuelnog kodiranja. Ove platforme omogućavaju korisnicima sa ograničenim tehničkim veštinama da razvijaju aplikacije pomoću prevlačenja i ispuštanja elemenata na platnu i konfigurisanjem njihovog ponašanja kroz prilagođene korisničke interfejse. Idealne su za brzu izradu prototipa i kreiranje jednostavnih aplikacija bez potrebe za kodiranjem.
24. **Od kojih faktora zavisi efekat primene RAD alata u razvoju softvera?**	Efekat primene RAD alata u razvoju softvera zavisi od faktora kao što su složenost aplikacija, kriva učenja, ograničeno prilagođavanje, integracija sa starim sistemima, proširljivost, dugotrajno održavanje, bezbednosne zabrinutosti, zaključavanje dobavljača i raspoloživost resursa.
25. **Šta u suštini predstavlja WORA ideja?**	WORA, akronim za "Write Once, Run Anywhere," označava ideju da se softverski kod može napisati jednom i zatim pokrenuti na različitim platformama bez potrebe za ponovnim prepravkom ili prilagođavanjem. Ova ideja je postala posebno poznata u kontekstu Java programskog jezika i njegove platforme.
26. **Gde se izvršava upravljani kod (managed code)?**	Upravljani kod (managed code) izvršava se u okviru .NET Framework-a, preciznije u izvršnom podsistemu poznatom kao CLR (Common Language Runtime).

27. U kojim oblicima je moguće kreirati Assembly (logičku jedinicu funkcionalnosti) pomoću .NET Framework-a?	Assembly (sklopovi) kreirani pomoću .NET Framework-a mogu biti ili izvršni programi (.EXE) ili ne-izvršne dinamičke biblioteke (.DLL) i predstavljaju gradivne blokove .NET aplikacija. Oni pružaju CLR izvršnom pod-sistemu informacije koje su mu potrebne kako bi on bio svestan implementacije tipova.
28. Koje vrste Assembly-ja u okviru .NET Framework-a postoje i gde su oni locirani?	U okviru .NET Framework-a postoje dve glavne vrste Assembly-ja: • **Privatni Assembly**: Ovo je .dll ili .exe koji je isključivo vlasništvo jedne aplikacije. Obično se čuva u osnovnom folderu aplikacije. • **Javni/Deljeni Assembly**: Ovo je .dll koji može koristiti više aplikacija istovremeno. Deljeni assembly se čuva u Globalnom kešu assembly-a (GAC), koji je folder na putanji C:\Windows\Assembly.
29. Koja je uloga JIT kompajlera u .NET Framework-u?	JIT (Just In Time) kompajler prevozi MSIL (Microsoft Intermediate Language) kod u izvršni kod koji operativni sistem može direktno izvršiti. Kompajliranje se odvija kada se assembly prvi put učita ili pozove. Nakon prvog prevođenja, izvršni kod se čuva, što omogućava brže izvršavanje prilikom sledećih poziva jer ponovo prevođenje nije potrebno.
30. Šta znači jezička neutralnost FCL (.NET Framework Class Library) biblioteka?	FCL (Framework Class Library) klase su jezički neutralne, što znači da se mogu koristiti iz bilo kog .NET jezika. To omogućava interoperabilnost između različitih .NET jezika, jer sve klase i funkcionalnosti u FCL-u mogu biti pristupane i korišćene bez obzira na jezik u kojem je aplikacija napisana.