19. KODIRANJE I TESTIRANJE

Programiranje i implementacija kao faza razvoja softvera koja mora biti prisutna u svim metodologijama razvoja se sastoji od sledećih aktivnosti:

- pisanja programskog koda,
- pisanja skripta,
- formiranja biblioteka klasa,
- izrade korisničkog interfejsa,
- kreiranja raznih vrsta datoteka određenog formata,
- dokumentovanje programa, izrada i pisanje uputstva za korišćenje programa.

Alati koji se koriste su: programski jezici treće (C++, C# , Java, Pyhton, Ruby, R, VisualBasic) i četvrte generacije (SQL), objektno orijentisane metode (OOM) pisanja koda, razvojna okruženja (Engl. "Integrated Development Environment") poput MS Visual Studio. NET, generatori aplikacija, kod editori – Visual Studio Code, Eclipse I sl., "GUI builders" - kreatori KI, DBMS sistemi za rukovanje bazama podataka (MySQL, Oracle, MS SQL Server, MS Access), radni okviri (Engl. "Frameworks") poput Entity Framework-a, Laravel-a, Symphony i sl.

Programski kod treba da bude generisan po zahtevima koji su opisani u dokumentu specifikacije zahteva korisnika ili potreba za funkcijama. Izvor za kodiranje mogu da budu, u skladu sa nekim metodologijama, poput OOR, modeli i njihovi dijagrami. Programeri moraju da prate uputstva za kodiranja (Engl. "coding style") koji su definisani standardima ili interno, od strane njihove organizacije i programskih alata kao što kompajleri, prevodioci, dibageri, koji se koriste za generisanje koda. Programski jezik kojim se izvodi projekat, bira se u odnosu na zahteve koji su definisani u dokumentu, a odnose se na performanse, pouzdanost, skalabilnost, modularnost i sigurnost softverskog rešenja koje se razvija.

Integracija i testiranje obuhvataju sledeće aktivnosti:

- Povezivanje delova programa, napisanih modula, klasa, datoteka i interfejsa.
- Pronalaženje grešaka ili nedostataka programa pre puštanja aplikacija u rad, tj. realizacije.
- Provera da li softver ispunjava tražene korisničke zahteve.

Alati koji se koriste su: softverski alati za automatizaciju testiranja (Postman, Cypress, Selenium i sl.). Testovi mogu biti: funkcionalni, nefunkcionalni, jedinični testovi, integracioni, regresioni i dr. Posebna pažnja se mora posvetiti testiranju bezbednosnih aspekata softvera (crna i bela kutija, testovi prodora i ranjivosti).

Testiranje softvera je proces evaluacije softvera koji se kreira tokom projekta razvoja softvera kako bi se utvrdilo da li je softver u skladu sa očekivanim zahtevima i potrebama korisnika, da li odgovara svojoj svrsi, kao i utvrđivanje da li postoje neki defekti u softveru. Loše funkcionisanje softvera sa defektima izaziva nezadovoljstvo korisnika i finansijske, materijalne gubitke, a ekstremno može čak i ljudske žrtve. Testiranje treba (ne po svim metodologijama) da prožima ceo životni ciklus razvoja, jer se u svakoj fazi razvoja vrši validacija i verifikacija napravljenog. Kao poseban korak, ova faza životnog ciklusa odnosi se na verifikaciju funkcionalnih i netehničkih zahteva softvera pre isporuke klijentima. Ova aktivnost se odnosi na testiranje softverskog proizvoda gde se izveštava o defektima proizvoda njihovo praćenje, popravljanje i ponovno testiranje, dok proizvod ne dostigne standarde kvaliteta koji su potrebni.

Povezanost modula i integracija softvera - od uspešnosti razdvajanja i povezivanja modula često presudno zavisi kvalitet projekta i uspešnost implementacije. Da bi mogao da se oceni kvalitet integracije, neophodno je proceniti kvalitet povezanosti, odnosno razdvojenost modula. Dve osnovne mere su kohezija i spregnutost:

- kohezija je stepen međusobne povezanosti elemenata jednog modula;
- spregnutost je stepen međusobne povezanosti elemenata različitih modula.

Dobro dizajniran sistem se odlikuje visokom kohezijom i niskom spregnutošću.

Kohezija u nekoj strukturnoj celini je visoka ako su svi njeni delovi neophodni i međusobno čvrsto povezani. Sa druge strane, spregnutost je niska ako su različite strukturne celine relativno slabo međusobno povezane. Koheziju i spregnutost se posmatraju na različitim nivoima funkcionalnog i strukturnog dekomponovanja sistema. Kohezija i spregnutost su apstraktne mere, koje se veoma često izražavaju samo opisno.

Visoka kohezija jednog modula znači da je su svi delovi jednog modula meusobno snažno povezani. Ako posmatramo jedan modul kao deo celine koji okuplja neke srodne elemente, onda je očekivano da ti elementi na neki način budu povezani, bilo da je ta povezanost funkcionalna (na primer, delovi jedne komponente sarađuju radi ostvarivanja funkcije komponente) ili logička (na primer, delovi jednog paketa su implementaciono međuzavisni).

Sa druge strane, niska kohezija u nekom modulu znači da pojedini delovi tog modula nisu međusobno povezani ili su vrlo slabo povezani. Niska kohezija je obično signal da dekompozicija možda nije sasvim dobro izvedena i da granice modula nisu ispravno određene. Ako u modulu možemo da prepoznamo manje grupe njegovih elemenata koji su međusobno snažno povezani a relativno slabo povezani sa ostatkom modula, onda bi verovatno trebalo nastaviti dekomponovanje, tj. podeliti modul na više manjih modula.

Spregnutost - niska spregnutost predstavlja nizak nivo međusobih zavisnosti između različitih modula. U slučaju dobre dekompozicije svaki modul predstavlja samostalnu celinu, koja koristi usluge drugog modula ili da pruža usluge drugom modulu. Cilj dobrog projektovanja je da se ostvari što bolje povezivanje modula koja se ostvaruje putem manjih i jednostavnijih interfejsa.

Visoka spregnutost predstavlja veoma snažnu povezanost nekog modula sa drugim modulima. Ako jedan modul intenzivno koristi usluge nekog drugog modula, ali kroz veoma jednostavan interfejs. Pak, ako se interfejs sastoji se od velikog broja funkcija, metoda ili klasa, ili pruža sasvim različite funkcije, onda je to znak da je lose urađena dekompozicija. Složen i obiman interfejs obično ukazuje na to da modul koji ima ima više odgovornosti ili nejasno definisanu odgovornost.

Visoka spregnutost između dva modula nam ukazuje da možda nije dobro postavljena granica između posmatranih modula, te bi možda trebalo da se radi o jednom modulu ili da neki delovi jednog od modula trebalo da pređu u drugi ili da se potpuno izmeni način dekomponovanja tog dela softvera.