1. UVOD

Razvoj informatike i njenih tehnologija je doveo do toga da su računari i programi uopšte postali deo svakodnevice za većinu ljudi. To se posebno odnosi na poslovnu sferu. U tom smislu pojavljuje se veliki broj metoda u oblasti softverskog inženjerstva, koje su namenjene za rešavanje problema razvoja softvera.

Razvoj softvera je složen proces, koji se sastoji iz više faza:

- zahtevanja određenih softverskih proizvoda,
- specifikacija i projektovanje,
- implementacija (primenjivanje),
- testiranje (testiranje se obično radi paralelno sa razvojem procesa, ali postoje i druge metode),
- pisanje dokumentacije (proizvod ne može biti iznet na tržište bez uputstva kako koristiti softver) i
- održavanje softvera (proizvod da bi radio bez poteškoća mora se održavati).

Veoma veliki uticaj ima kontrola kvaliteta u ovoj oblasti, a to potvrđuju i rezultati istraživanja nekoliko velikih i poznatih američkih istraživačkih organizacija: skoro pa više od dve trećine američkih softverskih projekata nikad ne uđe u upotrebu. Razlozi su sledeći: projekat ne završi u roku i budžetu, ne ispoštuje se ono što je klijent hteo ili je kvalitet samog izlaza ispod očekivanja (loše performanse, visoki troškovi održavanja i nadogradnje, preteško rukovanje, itd). Kvalitet samog softvera može se posmatrati na razne načine, ali izdvajamo dva najbitnija, a to su pomoću: efektivnosti i efikasnosti.

1. Efektivnost se odnosi da li softver pruža traženu funkcionalnost (da li radi ono što je klijentu potrebno).
2. Efikasnost, sa druge strane, podrazumeva da se funkcionalnost pruža na kvalitetan način (softver radi brzo, pouzdano – nema greške, lak je za održavanje, nadogradnju i sl.). U praksi se obično, osiguravanje efektivnosti vrši već u procesu sakupljanja korisničkih zahteva i opisa problema. Uglavnom se teži ka izradi određenih prototipova programa i prezentovanja krajnjim korisnicima da bi se odmah uočili neki nedostaci u funkcionalnosti ili u samom zahtevu. Još jedan od praktičnih postupaka za povećavanje efikasnosti je i tzv. “iterativni” razvoj softvera. Ceo razvoj se deli u nekoliko iteracija, a izlaz iz svake iteracije je program koji ima samo deo funkcionalnosti koji se odmah pušta u rad. Tako se, krajem svake iteracije, dodaje funkcionalnost sve dok se ne dobije gotova celina sa kompletnom funkcionalnošću – ceo program. Prednost ovakvog pristupa je u tome što se nedostaci u korisničkim zahtevima uočavaju i otklanjaju brzo i detaljno.

Efikasnost softvera je nešto što se prvenstveno postiže uvođenjem u praksi dokazanih postupaka pri projektovanju, implementaciji i testiranju softvera. Cilj je da se dobije softver koji je brz, pouzdan i lak za održavanje i nadogradnju. Korišćenje uzora, obrazaca (engl.pattern) u projektovanju softvera značajno doprinosi njegovoj robusnosti tako da su održavanje i nadogradnja relativno jednostavni. Sa druge strane, brzina i pouzdanost mogu da se potvrde i obezbede jedino detaljnim testiranjem.

Testiranje predstavlja najčešće zanemarivanu aktivnost u razvoju softvera. Ako se pogleda Larmanova metoda[1] razvoja softvera može se uočiti da pored opisa korisničkog zahteva, analize, projektovanja i implementacije, sadrži i testiranje kao fazu. Znači, prema Larmanu, testiranje se vrši posle implementacije (posle pisanja koda). Sličan pristup je opisan i u Jedinstevnom procesu razvoja softvera od strane Scott Ambler-a. Predstavnici pravca "ekstremnog programiranja" imaju drugačiji pristup: oni predlažu pisanje testova pre pisanja koda (engl.write tests before you write code).

U praksi se stvari odvijaju drugačije: testiranje se vrši retko - najčešće zbog vremenskih rokova, testiranje se vrši neorganizovano i ponekad se skroz izostavlja. Posledice su očigledne – dobija se softver loših performansi i pun grešaka, a gubi se novac i vreme za njegovu ispravku. Ipak, najveći gubitak je upravo u poverenju klijenata i njihovom zadovoljstvu.

Korišćenjem alata za testiranje i stvaranje "dobrih" navika kod programera da pišu jedinične (engl.unit) testove, stvara se put za prevazilaženje problema ove vrste. Moguće je praviti jeftiniji, a u isto vreme i brz i pouzdan softver. Dodatno vreme koje je utrošeno na testiranje će svakako biti vraćeno zbog smanjenog vremena potrebnog da se otklone greške ili povećaju performanse.

Najvažnije je, svakako, to da će klijenti biti zadovoljniji, i da će se troškovi izrade softvera verovatno smanjiti, jer je uvek jeftinije grešku ispraviti dok program još nije pušten u rad.

Nakon razvoja i testiranja softvera kao najbitnijih procesa razvoja aplikacije, retko ko kasnije obraća pažnju na samo održavanje softvera. Ovaj princip je od velikog značaja kako prvenstveno zbog ekonomičnosti (nepotrebno često podizanje – instaliranje komplikovanog i skupog sistema), ali i zbog same efikasnosti (brzine, odnosno bez potrebnog gubljenja vremena i pravljena tzv. novih backupa, pa premeštanja na nove servere, itd).

Glavni ciljevi ovog rada su:

- Predstaviti metode organizovanja razvoja, testiranja i održavanja softvera.
- Dati kratak pregled modela, metoda i aktivnosti životnog ciklusa softvera;
- Dati pregled .NET platforme;
- Prikazati osnovne procese održavanja softvera.

[1] **Craig Larman**: 1997. – „*Applying UML and Patterns"* – „Prihvatanje UML i obrazaca", Prentice Hall