2 - Problemi pri razvoju softvera

Ne paničite!
Daglas Adams

2.1 Problemi i neuspesi pri razvoju softvera

U razvoju softvera je problem sve ono što negativno utiče na razvojni proces i krajnji rezultat. Problemi mogu da otežavaju posao, da ga produžavaju, da mu povećavaju cenu, da nepotrebno zamaraju razvijaoce, da negativno utiču na odnose u razvojnom timu, da prouzrokuju niži kvalitet rezultata, pa čak i da u potpunosti obesmisle razvoj konkretnog proizvoda. Problemi vrlo retko mogu da se u potpunosti eliminišu, ali uz ulaganje dovoljno kvalitetnog rada na sprečavanju i prevazilaženju problema, njihove posledice mogu da se značajno umanje ili čak učine zanemarljivim.

Sa druge strane, ako se potencijalnim problemima ne posveti dovoljno pažnje, onda oni mogu da dovedu i do značajnih negativnih posledica po razvojni proces ili rezultat razvoja. Negativne posledice problema nazivamo neuspehom. Grubo posmatrano, neuspeh je svako nepovoljno odstupanje od plana. Uobičajeno je da se kao osnovno merilo neuspeha posmatra izgubljena materijalna vrednost. Ona se najčešće prepoznaje kroz prekoračenje predviđenog obima ulaganja sredstava ili uloženog vremena, ali ponekad i kroz posledice koje se odnose na čitav poslovni sistem. U skladu sa tim, obično se prepoznaju sledeće osnovne vrste neuspeha:

- prekoračenje troškova;
- prekoračenje vremenskih rokova;

- neupotrebljivost rezultata i
- odustajanje od projekta.

Osnovni oblik izgubljene materijalne vrednosti predstavlja neplanirano povećanje obima uloženih sredstava, tj. prekoračenje predviđenih troškova. Skoro svaki problem koji nastupi tokom razvoja, ima za posledicu potrebu da se deo nekog posla popravlja, ponavlja ili proširuje, a svako povećavanje obima posla podrazumeva i više radnih sati i više uloženih sredstava.

Povećanje obima posla nekada može da se kompenzuje povećanjem broja angažovanih učesnika u razvoju, tako da pored izgubljenih sredstava ne mora da dođe i do izgubljenog vremena i kašnjenja realizacije projekta. Ipak, vrlo često nije moguće da se dodatni radni sati uklope u planirani raspored poslova, pa se kao dodatna posledica pojavljuje i kašnjenje.

Pored već prepoznatog povećanja troškova usled angažovanja dodatne radne snage, izgubljeno vreme može da prouzrokuje dodatne troškove i iz raznih drugih poslovnih razloga, od produženog angažovanja prostora i sredstava za rad, pa do neugodnih posledica koje korisnik softvera može da ima zbog njegovog kašnjenja. Zakasnelo puštanje softvera u rad i nepredviđeno povećanje troškova ili redukovana upotrebljivost proizvoda mogu ne samo da naprave velike neprijatnosti, već i da u posebno ozbiljnim slučajevima proizvedu kritične posledice po ceo poslovni sistem za koji se softver razvija. Te posledice mogu da idu toliko daleko da obesmisle dalji rad na softveru, ili čak da dovedu u pitanje opstanak planiranog poslovnog poduhvata ili čak čitavog poslovnog sistema za koji se softver razvija.

Čak i kada je softver završen u skladu sa planovima (ili uz eventualno prihvatljivo probijanje rokova ili budžeta), može da se dogodi da ne može da se upotrebi i da je čitav razvoj bio uzaludan. Neupotrebljivost je obično posledica neispunjenosti nekog od ključnih funkcionalnih ili nefunkcionalnih zahteva ili prisustva veoma ozbiljnih bagova. Uobičajen uzrok takvih problema je loše praćenje projekta, tj. izostanak kvalitetnog nadzora i kontrole.

Najčešći oblici neupotrebljivosti razvijenog softvera obuhvataju:

- nesaglasnost zahteva po kojima je softver implementiran sa stvarnim potrebama;
- neispunjenost funkcionalnih zahteva;
- neispunjenost nefunkcionalnih zahteva;
- katastrofalni bagovi;
- nemogućnost da se sistem se pusti u rad;

- neupotrebljivost korisničkog interfejsa i
- neostvarivost procedure korišćenja u realnim uslovima.

Možda izgleda čudno, ali relativno često se dešava da je softver razvijen u potpunosti u skladu sa planovima i funkcionalnim zahtevima, da nema značajnih bagova, a da ipak ne može da se primeni. Da problem bude još neugodniji, do toga najčešće dolazi kod veoma velikih projekata. Razlog za to je obično u promenljivosti poslovnih procesa i dugotrajnosti razvoja složenih softverskih sistema – ako se softver razvija nekoliko godina, prema unapred definisanoj specifikaciji, sasvim je moguće da u međuvremenu te specifikacije prestanu da odgovaraju realnim potrebama poslovnog sistema i da softver samim tim više ne može da se primeni u planiranom obliku.

Loš korisnički interfejs i procedure korišćenja koje nisu prilagođene realnim praktičnim uslovima upotrebe takođe mogu da dovedu u pitanje upotrebljivost softvera. Loš korisnički interfejs po pravilu usporava upotrebu softvera i prouzrokuje veći zamor korisnika, a time i veći broj grešaka koje nastaju pri upotrebi softvera. Neprilagođene procedure korišćenja obično su posledica neispravnih pretpostavki o uslovima upotrebe softvera ili stručnosti i mogućnostima korisnika.

Jedan od značajnih uzroka neupotrebljivosti gotovih projekata predstavljaju propusti u vezi sa puštanjem sistema u rad. Složeni softverski sistemi se danas najčešće razvijaju sa ciljem da zamene neka postojeća rešenja. U takvim okolnostima puštanje novog sistema u rad nije sasvim jednostavno i može da zahteva prilično složene poslove prebacivanja podataka iz starog sistema u nov, isključivanje starog i uključivanje novog sistema po delovima i slične postupke. Međutim, u osetljivim radnim okruženjima dodatnu otežavajuću okolnost može da predstavlja potreba da poslovanje ne sme da bude prekinuto zbog puštanja novog sistema u rad. Ako pri planiranju novog sistema nije dovoljno dobro i detaljno isplaniran postupak puštanja u rad, onda u toj fazi može da dođe do značajnih problema, koji često zahtevaju veće modifikacije softvera, nekada i toliko obuhvatne da dovedu u pitanje smisao celog projekta.

2.2 Primeri neuspeha

Neuspesi nisu lepa tema, a oni koji ih dožive ili čak prouzrokuju, ne ponose se mnogo njima, pa se zato o neuspesima ne govori često. Ipak, s vremena na vreme neki od neuspešnih slučajeva razvoja softvera dospe u naučnu ili stručnu literaturu, ili bude predstavljen javnosti putem medija. Ovde ćemo istaći neke od zanimljivih nalaza istraživanja, koje je sprovela Stendiš grupa 1994. godine [*Standish Group 1994*], a čitaocima sugerišemo da to istraživanje pročitaju u celosti:

- Prosečno godišnje ulaganje u razvoj primena IT u SAD je iznad $250.000.000.000 i to u prosečno 175.000 IT projekata.
- Oko 31% projekata je obustavljeno pre završetka.
- Oko 52.7% „uspešnih" projekata je proizvelo prekoračenje početnog budžeta za oko 89%.
- Dodatni troškovi, koji ne čine neposredna ulaganja u razvoj, već su posledica kašnjenja ili neuspeha projekta, procenjuju se na bilione dolara (hiljade milijardi).
- Procenjuje se da je utrošeno oko $81.000.000.000 na obustavljene projekte.
- Procenjuje se da je utrošeno oko $59.000.000.000 na dodatne neplanirane troškove projekata koji su dovršeni.
- Samo 16.2% projekata je završeno na vreme i bez probijanja planiranog budžeta.
- Samo 9% projekata u velikim kompanijama je završeno na vreme i bez probijanja planiranog budžeta.

Jedna novija analiza [*Standish Group 2015*] nam sugeriše da je uspeh projekata (završetak uz poštovanje rokova i plana troškova) obrnuto proporcionalan njihovoj veličini. U studiji se ispostavilo da je od posmatranih projekata bilo uspešno tek oko 58% malih agilnih i 44% malih klasičnih projekata, kao i tek oko 18% velikih agilnih i 3% velikih klasičnih projekata. Sličan je i odnos složenosti i uspešnosti.

U skladu s tim, prepoznaju se tzv. *pobednički uslovi*, kao mali projekti sa iskusnim razvojnim timovima i savremenim razvojnim metodima, ali i *gubitnički uslovi*, kao veliki projekti sa neiskusnim timovima i prevaziđenim razvojnim metodima.

Istraživanje o razlozima neuspešnosti, sprovedeno na 52 IT projekta u oblasti zdravstva u Saudijskoj Arabiji [*Abouzahra 2011*] pokazuje da su najčešći uzroci neuspeha:

- nedefinisani okviri projekta (44%);
- neprepoznati rizici (30%);
- neprepoznati ulagači (14%);
- loša komunikacija (7%) i
- drugi razlozi (5%).

Interesantni primeri ekstremnih slučajeva neuspeha se navode u radu [*Charette 2005*]:

- Kompanija *FoxMeyer Drug Co.* iz Teksasa je 1996. godine bankrotirala zbog loše implementacije sistema za planiranje resursa. Vrednost kompanije je neposredno pre toga bila procenjena na $5.000.000.000.
- Federalna uprava za vazduhoplovstvo SAD je radila na razvoju sistema za kontrolu letova od 1981. do 1994. godine, kada se odustalo od projekta posle ulaganja od $2.600.000.000, što je bilo trostruko više od planiranih troškova. Ukupni gubici zbog neostvarenog unapređenja poslovanja se procenjuju na više od $50.000.000.000.

U istom radu je napravljena i dobra analiza razloga ovih (i drugih) neuspeha, pa ga toplo preporučujemo čitaocima.

2.3 Uzroci problema

Uzroci problema se obično dele na uzroke na strani klijenta (tj. naručioca i budućeg korisnika softvera), uzroke na strani razvijalaca softvera i uzroke koji se odnose na obe strane. Ovakvu podelu pravimo samo radi lakšeg sagledavanja uzroka i njihovih međusobnih odnosa, a ne i da bismo sugerisali ko je odgovoran za rešavanje problema.

Iako sami uzroci problema mogu da se nalaze na strani klijenta, pa čak može da bude sasvim očigledno da je klijent u potpunosti odgovoran za njihovo nastajanje, ipak moramo da imamo u vidu da je odgovornost za uočavanje i prevazilaženje problema nastalih tokom razvoja softvera praktično uvek na strani razvijalaca i posebno na strani rukovodilaca razvojnog tima. Naime, ako je neke probleme klijent sam uočio i rešio, onda razvojni tim obično nije ni imao priliku da prepozna takve probleme i njihove posledice. Nasuprot tome, ako neke probleme klijent nije primetio, onda će upravo razvojni tim biti u prilici da se sa njima suoči. Zbog toga razvojni tim mora uvek da bude na oprezu i da se trudi da blagovremeno uoči probleme na strani klijenta, pre nego što njihove posledice počnu da se u značajnijem obimu odražavaju na razvojni proces.

Uzroci na strani klijenta

Najčešći uzroci problema koji se prepoznaju na strani klijenta su:

- nerealni ili nejasni ciljevi projekta;
- neusklađenost ciljeva i strategije;
- politika ulagača;
- komercijalni pritisak i
- otpor korisnika prema primeni novog softvera.

Nerealni i nejasni ciljevi projekta se odnose na sve one slučajeve kada ne postoji prepoznat cilj koji je ostvariv u rokovima i sa troškovima koje klijent može i želi da podnese. Savremene informacione tehnologije su svojom sveprisutnošću stvorile u delu šire javnosti utisak da sve može da se napravi i da svaki cilj može da se ostvari. To može da podstakne potencijalne investitore da pokrenu razvojne projekte bez dovoljno dobro sprovedene prethodne razrade i bez jasno definisanih ciljeva.

Osnovni problem sa nepreciznim i nerealnim ciljevima je to što oni imaju tendenciju da se tokom vremena menjaju. Njihovim postepenim menjanjem na kraju može da se dođe do jasnijih i realnijih ciljeva, ali je problem što promene ciljeva obično povlače za sobom velike promene funkcionalnih zahteva. Svaka promena zahteva u toku razvoja dovodi do postavljanja pitanja da li su time poslovi koji su već obavljeni i rezultati rada koji su do tada ostvareni možda postali pogrešni ili nepotrebni, pa tako preti da praktično otpiše deo već uloženih sredstava i proizvede značajne gubitke.

Sličan problem nastaje i ako na strani klijenta ne postoji jedinstven stav o strategiji razvoja, pa se tako dešava da jedan sektor kompanije postavi vrlo jasne i utemeljene ciljeve razvojnog IT projekta (bar na lokalnom nivou, iz ugla tog sektora) i uloži sredstva da bi se on ostvario, a da zatim na višem nivou odlučivanja dođe do nekih strateških odluka koje potpuno obesmisle čitav projekat. Takvi problemi nisu retkost u poslovnim okruženjima u kojima nisu dovoljno jasno definisane nadležnosti i ne postoji dobra komunikacija između delova kompanije. Tu možemo da primetimo vrlo neprijatan paradoks – takvo stanje obično nastupa u brzo rastućim kompanijama, a one čine možda najvažnije investitore u oblasti informacionih tehnologija.

Neujednačen stav o projektu među različitim subjektima na strani klijenta može da proizvede različite vrste problema, od nepreciznih specifikacija i čestih promena zahteva, pa sve do neupotrebljivosti implementiranog softvera. Neujednačen stav može da bude posledica, između ostalog, neposvećenosti rukovodilaca, nedovoljne komunikacije sa budućim korisnicima ili veštačke integracije više potencijalnih projekata u jedan (obično radi zamišljene uštede), kao i nekih drugih razloga. Sve ove probleme prepoznajemo kao probleme koji potiču iz politike ulagača.

U vezi sa prethodnim je i potencijalni otpor dela budućih korisnika prema upotrebi razvijenog softvera. Takav otpor je često posledica predrasuda, ali u nekim slučajevima predstavlja i racionalan ili iracionalan otpor prema promenama koje na neki način narušavaju položaj zaposlenih ili uslove rada. Uopšteno posmatrano, svako uvođenje informacionih tehnologija u poslovne procese dovodi do bar dve stvari koje zaposleni ne vole: do veće kontrole (najpre do boljeg uvida u odvijanje aktivnosti, a time i do veće kontrole zaposlenih) i do promena u načinu obavljanja poslova (a time i neophodnosti učenja novih postupaka). U dobro organizovanim sredinama taj otpor može da se usmeri u pozitivnom smeru uključivanjem šireg

skupa zaposlenih u proces planiranja, tako da se i bolja kontrola i promene u načinu rada prepoznaju kao unapređenje uslova rada, a ne kao njihovo narušavanje.

Svaki komercijalni razvojni projekat, pa i ulaganje u razvoj softvera, pokreće se da bi se ostvario profit. Iz različitih razloga (na primer, zbog finansijskog neuspeha nekog drugog projekta, zbog nepoverenja u razvojni tim i mnogo drugih razloga) u toku razvoja može da se pojavi pritisak da se planirani profit, ili bar njegov deo, ostvari što pre. U takvim okolnostima vrlo često se donose pogrešne odluke, poput menjanja redosleda implementiranja delova softvera, nerealnog smanjivanja rokova, smanjivanja ili potpunog preskakanja nekih vidova kontrole kvaliteta i drugo. Naravno, najbolje je da se projektni tim odupire takvim pritiscima, ali u realnim uslovima to često nije moguće, pa se reakcija razvojnog tima često svodi na pokušaje kontrole štete, tj. pokušaje da se insistira na doslednoj implementaciji najvažnijih delova softvera, a da se uštede (a time i eventualni problemi) ograniče na manje osetljive delove softvera.

Uzroci na strani razvojnog tima

Kada se posmatraju uzroci problema na strani razvijalaca softvera, njihova zajednička karakteristika je da su praktično u potpunosti u domenu rukovođenja razvojnim procesom. U najčešće uzroke se ubrajaju:

- slabo vođenje projekta;
- neprecizna procena potrebnih resursa;
- slabo izveštavanje o stanju projekta;
- neupravljani rizici;
- upotreba „nezrelih“ tehnologija;
- nesposobnost da se iznese složenost projekta i
- tok razvoja bez čvrstih principa i pravila.

Svaki od ovih uzroka nastaje kao posledica nestručnosti ili lošeg rada rukovodilaca razvojnog tima. Doduše, u nekim slučajevima (na primer, u slučaju upotrebe nezrele tehnologije koja je nametnuta od strane klijenta) prostor za reagovanje može objektivno da bude sužen, pa je onda i lična odgovornost nešto niža.

Najčešći uzroci problema koji se nalaze i na strani klijenta i na strani razvijalaca softvera su:

- slaba komunikacija između klijenta, razvijaoca i korisnika;
- nepoverenje između klijenta i razvijaoca i
- loše definisani zahtevi

Kao što smo videli, svaki od uzroka na strani razvojnog tima ima korene među rukovodiocima razvoja, ali i ostali problemi često mogu da se na vreme prepoznaju i otklone od strane rukovodioca razvoja, pa zato svaka priča o problemima obično počinje i završava se u okviru rukovodećeg dela razvojnog tima. U praksi, iza svakog neuspeha obično stoji više uzroka. Pojedinačni uzroci problema obično nemaju kapacitet da upropaste projekat, ali ako se nezgodno složi više uzroka, onda ishodi mogu da budu veoma nepovoljni. Tim pre je važno da se praćenju projekta i sagledavanju mogućih rizika posvećuje puna pažnja tokom čitavog trajanja projekta. Svako kašnjenje u uočavanju problema ili reagovanju na njih može da ima loše posledice.

Loše planiranje

Jedan od najčešćih uzroka problema i neuspeha je loše planiranje. Loše planiranje može da nastupi u bilo kojoj fazi razvoja, pa se tako loše planiranje u početnim fazama razvoja obično ispoljava u obliku nerealnih ili nejasnih ciljeva projekta ili loše definisanih zahteva, a u kasnijim fazama kao slabo vođenje projekta ili neprecizna procena potrebnih resursa. Loše planiranje se obično izjednačava sa nedovoljnim planiranjem, ali to je ozbiljan previd, zato što je nedovoljno planiranje samo jedan od oblika lošeg planiranja – drugi oblik, a koji može da ima jednako neugodne posledice, jeste preterano planiranje.

Nedovoljno planiranje je obično posledica nedovoljno dobro sprovedene analize zadatka ili loše ili neprecizno definisanih zahteva. Da bi mogao da se napravi dobar plan razvoja, neophodno je da se prethodno dobro analizira problem koji pokušava da se reši razvijanim softverom. Analiza problema mora da bude dovoljno široka i duboka da pruži odgovore na sva pitanja iz domena problema, koji su potrebni da bi se napravili projekat softvera i plan razvoja. Posledice nedovoljno dobre analize su neuočavanje nekih značajnih elemenata domena koji utiču na rešavanje problema, previđanje nekih poslova koje je neophodno uraditi ili nekih koraka kojima se obezbeđuju neophodne ulazne informacije, neprepoznavanje specifičnih vrsta subjekata koji će imati različita očekivanja od softvera i drugo. Kada projekat softvera nastane na osnovu nedovoljno dobre analize, obično iz njega izostanu neki neophodni koncepti ili komponente, ili neki zastupljeni koncepti ili komponente počivaju na nižem nivou apstrakcije nego što je potrebno, što kasnije otežava dogradnju izostavljenih funkcija.

Neprecizno definisani zahtevi su obično posledica „podrazumevanja trivijalnih stvari“ od strane projektanta. Moramo da priznamo da navođenje trivijalnih stvari zaista može da nepotrebno optereti projekat, ali i da primetimo da je procenjivanje „trivijalnosti“ veoma subjektivno i da može da odvede projekat u lošem smeru. Čak i ako je projektant možda već pravio mnogo sličnih projekata, svejedno mora da ima u vidu da neki od razvijalaca možda nemaju takvo iskustvo i da zbog toga ne mogu da razumeju (odnosno da podrazumevaju) ono što nije napisano u okviru specifikacije

zahteva. Projektna dokumentacija je sredstvo za komunikaciju članova razvojnog tima i svaka nepotpunost te dokumentacije preti da dovede do nerazumevanja između članova tima, a samim tim i do različitih problema koji iz tog nerazumevanja mogu da proisteknu.

U najvažnije posledice nedovoljnog planiranja spadaju nesrazmerno veliki broj naknadnih korekcija zahteva, slaba upotrebljivost rešenja i probijanje rokova i troškova. Ako se slabosti u zahtevima ne uoče na vreme, onda može da se dobije softver koji nije funkcionalan i koji u praksi ne može da se upotrebi u punoj meri i u planiranom obliku. Sa druge strane, ako se uoče, onda će biti neophodne naknadne korekcije. Naknadne promene ili dopune zahteva obično proizvode dodatne poslove, zato što novonastalim izmenama mora da se prilagođava sve što je počivalo na prethodnim verzijama zahteva, uključujući projekte, planove, a često i implementacije nekih delova softvera. To dalje utiče na povećavanje troškova i trajanja razvoja.

Možda izgleda paradoksalno, ali relativno čest uzrok problema u razvoju softvera je neki vid preteranog planiranja. Preterano planiranje se obično ogleda kroz:

- preširoko i nekoncentrisano ulaženje u projekat;
- suviše duboku i obimnu analizu sa ranim detaljnim projektovanjem;
- preširoko ulaženje u implementaciju i
- preambicioznost, nesrazmernu realnim mogućnostima (tzv. pozlaćivanje).

Preterano planiranje može da proizvede projektnu dokumentaciju velikog obima, na primer i do nekoliko hiljada stranica. Veliki obim dokumentacije otežava uočavanje važnih informacija i lako može da se dogodi da se zbog toga naprave neki suštinski propusti. Neke od najčešćih posledica preteranog planiranja su kašnjenje u uočavanju propusta i otežana tranzicija. Što je neka projektna dokumentacija veća, to je teže sagledati sve njene aspekte. Zbog velikog obima dokumentacije razvijaocima je veoma teško da pri razvoju nekih elemenata uvek uzimaju u razmatranje i sve ostale povezane elemente, pa je im je zbog toga teško i da ispravno implementiraju i da ispravno testiraju implementirano. Zbog toga se dešava da se čak i neki konceptualni propusti uočavaju tek pri testiranju gotovog proizvoda, što je obično suviše kasno da bi popravke mogle da se uklope u rokove i budžet.

U vezi sa tim je i činjenica da je složenije softverske sisteme teže pustiti u rad. Tranzicija je uvek složena, pa je posebno važno da je ne činimo još složenijom ako to nije neophodno. Pozlaćivanjem projekta, njegovim nepotrebnim širenjem i implementacijom elemenata koji nisu posebno važni za korisnika ne gubi se samo na

vremenu i sredstvima za razvoj, nego često i još mnogo više na vremenu i sredstvima za puštanje sistema u rad.

2.4 Prevencija problema

Analize najčešćih problema i uzroka njihovog nastajanja pokazuju da u najvažnije uzroke problema spadaju loša komunikacija razvijalaca i klijenata, loše planiranje i izmene koje nastaju nakon projektovanja a pre dovršavanja softvera. Savremene razvojne metodologije pokušavaju da se suoče upravo sa tim problemima. One počivaju na konceptima i tehnikama koje omogućavaju pouzdaniji razvoj i obezbeđuju manji rizik nastajanja problema. U nastavku ovog poglavlja sagledaćemo neke od važnijih koncepata, dok ćemo izabrane tehnike razvoja softvera predstaviti u narednim poglavljima.

Pojačana komunikacija među subjektima

Procenjeno je da je jedan od osnovnih problema klasičnih metodologija relativno nizak nivo komunikacije između svih subjekata uključenih u projekat razvoja softvera, a prvenstveno između klijenta i razvojnog tima. Veoma često se dešavalo da je ta komunikacija bila intenzivna na početku projekta, u vreme planiranja i projektovanja, a zatim je praktično skoro potpuno izostajala tokom dugačkog perioda razvoja. Česta posledica je bilo značajno odstupanje izvedenog projekta od onoga što je klijent zaista želeo. To je moglo da se desi iz više razloga, a najčešće zato što bi se u međuvremenu promenila realnost poslovanja klijenta, a time i njegove potrebe u odnosu na projekat, ili bi neki segment projekta bio pogrešno protumačen od strane projektanata ili implementatora, a da klijent to nije shvatio u fazama planiranja ili ugovaranja posla.

Zato jedan od najvažnijih koncepata savremenog razvoja softvera predstavlja insistiranje na intenzivnoj i redovnoj komunikaciji među subjektima koji na bilo koji način učestvuju u razvoju softvera, a pre svega između klijenta i razvojnog tima. Cilj pojačavanja komunikacije je prevencija problema koji nastaju usled izostanka komunikacije, ali i stvaranje uslova za kvalitetniju saradnju i bolje rezultate rada.

Osnovna dobit od redovne komunikacije između klijenta i razvojnog tima je u blagovremenom uočavanju eventualnih odstupanja od klijentskih potreba i reagovanju na njih. Zaista, ako klijent u regularnim intervalima razmatra stanje razvoja softvera, onda može da na vreme doprinese ne samo uočavanju grešaka nego i eventualnom oblikovanju još boljih rešenja za neke segmente posla. Stalna komunikacija dodatno doprinosi porastu međusobnog poverenja između klijenta i razvojnog tima. Sa jedne strane, ako klijent vidi da se razvojni proces odvija po planu, onda će biti manje zabrinut za svoju investiciju, imaće više poverenja u razvijaoce i vršiće manji pritisak na njih. Sa druge strane, ako razvijaoci vide da je klijent zadovoljan, onda će moći da rade sa manje rizika, u rasterećenijem okruženju.

Na osnovu informacija koje dobiju od različitih subjekata sa klijentske strane, članovi razvojnog tima mogu da steknu tačniju sliku o potrebnim ciljevima, čime se smanjuje rizik razvoja neupotrebljivog rešenja. Slično, ako neka strana nije zadovoljna, bolje je da se to ustanovi i reši na vreme, nego da se nezadovoljstvo prikuplja do tačke pucanja projekta po šavovima.

Pojačana komunikacija podrazumeva i intenzivniju razmenu informacija između svih članova razvojnog tima, kao i između različitih timova koji učestvuju u razvoju. Jedan od ciljeva te komunikacije je distribuiranje informacija među članovima tima, tako da više članova tima može da se uključi u rešavanje eventualnih problema ali i da se lakše nadoknadi nečije eventualno odsustvo.

Stalna i intenzivna komunikacija nosi i neke rizike. Velika količina dostupnih informacija može da ima upravo suprotne efekte i da klijentu oteža sagledavanje stanja projekta, umesto da ga učini lakšim i jasnijim.

Takođe, pružanjem prilike da neprekidno utiče na tok i način rada, možemo nepripremljenog klijenta da dovedemo u situaciju da neracionalno velikim brojem sugestija za menjanje i unapređivanje planova i rezultata rada napravi suprotne efekte. Neodmereno i nestručno učešće klijenta može da dovede do usporavanja projekta i putem eventualno preteranog planirana ili *naduvavanja* čitavog projekta ili pojedinačnih koraka projekta. Poseban problem mogu da predstavljaju tzv. neprekidni nizovi izmena, kada klijent stalno iznosi nove zahteve za izmene, ne čekajući ni da prethodno izneseni zahtevi budu implementirani. Nasuprot tome, posvećivanjem prevelike pažnje stavovima konzervativnijih subjekata, koji su navikli na postojeće procese i ne sagledavaju dobro planirane izmene, može da se smanji obim suštinskih funkcionalnih izmena u okviru pojedinačnih razvojnog koraka, ali tako da se time nepotrebno uspori razvoj i poveća potreban broj razvojnih koraka na putu do finalnog proizvoda.

Uključivanjem budućih korisnika u razvojni proces omogućava se blagovremeno uočavanje i popravljanje eventualnih slabosti korisničkog interfejsa ili zamišljenih procedura upotrebe softvera. Budući korisnici se upoznaju sa novim softverom u njegovim ranim fazama i tako se postupno obučavaju za njegovu upotrebu. Na taj način klijent praktično od prvog dana razvoja može da započne na pripremama za upotrebu novog sistema.

Jedan vid pojačavanja komunikacije sa klijentom predstavlja pravljenje *prototipova*. Prototip obično predstavlja kostur rešenja iz koga, kroz relativno primitivno izveden korisnički interfejs i simulirano izvođenje poslova, može da se sagleda kako će softver ili deo softvera da funkcioniše kada bude završen. Iz ugla razvojnog tima prototip nekada može da izgleda kao suvišan posao, ali je njegova uloga u razvojnom procesu često nezamenljiva. On obično pruža samo *iluziju* upotrebe, ali u dovoljnoj meri da budući korisnici i klijenti mogu da steknu utisak o načinu funkcionisanja budućeg softvera, neposrednije i tačnije nego što bi to mogli

samo na osnovu projektne dokumentacije, koju obično i ne razumeju u potpunosti zato što nemaju dovoljno tehničkih znanja. Kada vide prototip i „osete pod prstima" kako će softver da radi, klijenti će moći da uspešnije uoče eventualne nedostatke i da konkretnije iznesu eventualne primedbe. Na taj način se smanjuje rizik od donošenja pogrešnih odluka u ranim fazama razvoja.

Ipak, i prototipovi nose neke rizike. Oni obično predstavljaju samo funkcionalne aspekte korisničkog interfejsa a ne i unutrašnju strukturu sistema koji se razvija. To može da ima za posledicu da se klijent fokusira na korisnički interfejs i možda zapostavi neke važnije aspekte planiranog softvera. Istovremeno, prototipovi koji nisu dovoljno široki, mogu da prikriju nedostatke u delovima softvera koji njima nije predstavljen. Takođe, preterano posvećivanje prototipovima, na primer sa težnjom da izgledaju i vizualno i sadržajno kao gotovo rešenje, može da uzme nepotrebno mnogo vremena..

Savremene razvojne metodologije zahtevaju stalnu i temeljnu kontrolu tokom čitavog razvojnog procesa. Što je neki projekat veći i što više različitih ljudi učestvuje u razvoju, to je važnije da se svaki korak temeljno proverava. Pored proveravanja ispravnosti urađenih delova posla, u kontrolne procedure spada i dobro praćenje i izveštavanje o stanju projekta. Blagovremeno i dovoljno iscrpno izveštavanje o stanju projekta može da bude presudno za blagovremeno uočavanje propusta i da značajno smanji troškove zbog pravljenja odgovarajućih korekcija. I to je vid komunikacije među učesnicima u razvoju softvera.

Iterativni razvoj

Jedan od važnih koncepata savremenih razvojnih metodologija je *iterativni* ili *inkrementalni razvoj*[4]. Klasične razvojne metodologije su se odlikovale posmatranjem razvoja softvera kao celovitog proizvoda. Neke od njih su prepoznavale potrebu da se projekat razvija po delovima, ali je svejedno bilo uobičajeno da se u početnim fazama posla do detalja razmatra i projektuje čitav proizvod. Iterativni razvoj, umesto toga, promoviše podelu razvojnog procesa na više faza i podelu posla na više manjih celina, koje se i razmatraju i implementiraju tek u odgovarajućim fazama.

Iterativni pristup razvoju softvera se razlikuje od, na primer, građenja zgrade, gde se najpre sve detaljno isprojektuje, a onda se sve to i implementira. Umesto toga, posao se prvo deli na manje celine i onda se jedan po jedan deo softvera najpre projektuje pa onda implementira. Na početku projekta se, naravno, i dalje sagledavaju potrebni delovi softvera, kao i poslovi koje je potrebno uraditi. Razlika je

[4] Uobičajen je termin *inkrementalni razvoj*, ali on implicira inkrementalnu prirodu faza razvoja, tj. jedan specifičan pristup planiranju i ostvarivanju iteracija. U daljem tekstu ćemo videti da to nije jedini pristup, pa je bolje da se koristi opštiji termin *iterativni razvoj*.

u tome što se na početku posla prave samo relativno grube procene, dok se detaljno analiziranje i projektovanje svakog dela radi tek u odgovarajućoj fazi. U tom smislu, razvoj softvera više liči na izgradnju jednog velikog naselja nego na izgradnju jedne zgrade – iako na početku imamo nekakvu veliku sliku o tome šta će i kako biti napravljeno, celine se rade jedna po jedna, pa zato pojedinačni delovi lakše trpe veće izmene. Na taj način se skraćuje vreme koje protekne od planiranja do završetka svakog pojedinačnog dela, a time se smanjuje i rizik da u toku implementacije jednog dela dođe do većih izmena u specifikaciji. Naravno, manje celine je mnogo lakše planirati i praviti, pa se dobija i na smanjenju troškova.

Svaka iteracija može da bude inkrementalna, evolutivna ili kombinovana. *Inkrementalna iteracija* dodaje softveru novi deo (komponentu, inkrement). Nizom inkrementalnih iteracija softver postepeno raste od jedne komponente do celog proizvoda, nalik na pravljenje i slaganje kockica.

Nasuprot tome, *evolutivna iteracija* ne dodaje novi deo, već unapređuje neki već postojeći deo (komponentu) softvera. Nizom evolutivnih iteracija softver raste tako što se postepeno povećavaju funkcionalnost i složenost postojećih komponenti.

Kombinovana iteracija dodaje nove delove i istovremeno unapređuje neke već postojeće delove softvera. Inkrementalne iteracije takođe često moraju da obuhvate neka manja prilagođavanja postojećih delova programa, pa se zato za iteraciju kaže da je kombinovana samo ako pravi relativno značajne izmene u postojećem kodu.

Primarna očekivana korist od iterativnog pristupa je u smanjivanju trajanja razvojnog ciklusa i intenziviranju komunikacije između klijenta i razvojnog tima. Sekundarna dobit je povećana preciznost napravljenih procena, zato što se detaljne procene prave u manjem obimu i na kraće rokove, pa su zato i merodavnije. Uspostavljanje ritma redovnog isporučivanja novih verzija ili delova sistema omogućava blisku saradnju sa klijentom i brže uspostavljanje visokog stepena poverenja između klijenta i razvijalaca. Omogućavaju se i redovnija kontrola kvaliteta i postepeno privikavanje korisnika na nove elemente sistema.

Iterativni razvoj ima i nekih slabosti. Najpre, ako se u prvim iteracijama potpuno ili delimično zanemare predstojeće iteracije, onda postoji rizik da će u narednim iteracijama biti neophodno da se preduzmu nešto veće izmene već implementiranih delova softvera. Sa druge strane, ako se u prvim iteracijama uzmu u obzir i sve predstojeće iteracije, onda postoji rizik da se složenost tih prvih iteracija približi složenosti čitavog sistema („naduvavanje koraka“), čime ovakav pristup gubi smisao. Nezgodna karakteristika iterativnog razvoja je i to što često nije moguće da se unapred tačno sagledaju broj, cena i ukupno trajanje svih iteracija. Takođe, ako se vrši preterano učestalo isporučivanje novih iteracija, onda to može i da podigne cenu projekta, zato što svaka isporuka zahteva dodatno vreme i dodatni rad.

Specifičan oblik iterativnog razvoja je da se iteracije ne planiraju prema poslovima već prema rokovima. Određivanje koraka prema rokovima počinje od ustanovljavanja koliko dugo će neka iteracija da traje i koliko ljudi će na njoj da radi. Zatim se odabiru poslovi za tu iteraciju, prema značaju, mogućnosti povezivanja sa već implementiranim delovima, ali pre svega prema kapacitetu te iteracije. Više savremenih agilnih metodologija propagira ovakav razvoj. Motivacija počiva na pretpostavci da ovakav pristup planiranju iteracija dodatno uvećava kvalitet iteracija, kako kroz preciznije pridržavanje planova rokova i sredstava tako i kroz promovisanje još intenzivnije komunikacije između razvijalaca i klijenata.

Određivanje koraka prema rokovima ima i neke potencijalno nezgodne karakteristike, koje moramo da imamo u vidu. Uspostavljanje tesnih okvira inkrementalnog koraka može da oteža pojedine korake i da podigne ukupnu cenu i trajanje razvoja. Neki elementi sistema ne mogu da se prirodno podele na različite korake, pa je zato nekada isplativije da se relaksiraju (produže) neke iteracije nego da se nešto veći poslovi veštački dele na više iteracija.

U prvim iteracijama se obično biraju poslovi koji donose veću dobit klijentu, pa kako se razvoj bliži kraju, nastupa rizik da se razvoj prekine pre nego što se implementiraju i poslovi „čija je cena veća od dobiti“ iako su oni možda presudno značajni za uspešno funkcionisanje softverskog sistema kao celine.

Objektna orijentacija

Savremene razvojne metodologije su po pravilu objektno-orijentisane (OO), što znači da se u prvi plan stavljaju objekti, a ne procesi, koji su uobičajeno zauzimali centralno mesto kod klasičnih metodologija. Osnovna dobit od fokusiranja na objekte je u povećanoj stabilnosti modela i proizvoda. Suština je u tome da promene u načinu poslovanja mnogo češće imaju za posledicu promene u postojećim procesima (postupcima), pa čak i nestajanje nekih i pojavljivanje potpuno novih procesa, nego što je to slučaj sa objektima.

Značajan faktor stabilnosti OO modela predstavlja enkapsulacija. Sakrivanjem implementacije iza interfejsa se omogućava da većina promena ima sasvim lokalnu prirodu. Naravno, ako se procesi implementiraju kao metodi nekih klasa objekata, onda će te klase objekata morati da se prilagođavaju promenama, ali se ispostavlja da pri tome postojeće klase nestaju i nove nastaju relativno retko (bar u odnosu na procese), a čak i pri njihovom modifikovanju, u ključnim odnosima među klasama dolazi do sasvim umerenog broja promena. Zbog toga, ako projekat i model softvera ili dela softvera zasnujemo na objektima i odnosima među objektima, sva je prilika da će najvažniji i najosetljiviji elementi projekta morati da se menjaju ređe nego što bi to bio slučaj da smo ih zasnovali na procesima.

Dodatna dobit od upotrebe OO koncepata je da se takav pristup lakše prilagođava iterativnom razvoju. Kada se u ranim fazama planiranja više pažnje

posveti objektima, a tek u kasnijim fazama procesima, onda se smanjuje i rizik od pogrešnih odluka u ranim fazama razvoja.

Osnovni rizik pri fokusiranju na objekte je eventualno potpuno zanemarivanje razmatranja procesa u ranim fazama razvoja, što može da vodi pogrešnoj arhitekturi sistema, koja se kasnije veoma teško (tj. skupo) menja. Sa druge strane, detaljno razmatranje procesa u početnim koracima modeliranja sistema preti da dovede do „naduvavanja" tih koraka. Zbog toga je neophodno da se u početnim fazama modeliranja sistema procesi razmatraju i vrlo pažljivo i odmereno.

2.5 Umesto zaključka

Savremeni metodi razvoja softvera su donekle umanjile učestalost neuspeha, ali još uvek se veliki broj softverskih projekata ne završava sa planiranim pozitivnim rezultatima. Zbog toga je veoma važno da se sprovode istraživanja koja su usmerena na što bolje prepoznavanje problema i pronalaženje boljih, efikasnijih i pouzdanijih metoda za njihovu prevenciju.

Zainteresovanima za ovu oblast preporučujemo radove o razlozima neuspeha u razvoju softvera, kao i knjige koje se bave metodološkim pitanjima razvoja informacionih sistema, u kojima se često posvećuje pažnja i uzrocima neuspeha i načinima njihovog sprečavanja. Pored izvora na koje je već referisano u tekstu [*Stangish Group 1994, 2015; Abouzahra 2011; Charette 2005*] čitaoce možemo da uputimo i na [*Avison 2003; Al-Ahmad 2009; Boehm 1991; Fortune 2005*].

Praktično svaka od savremenih tehnika razvoja softvera je oblikovana radi prevazilaženja ili smanjivanja rizika od nastajanja neke vrste problema sa kojima se svakodnevno suočavamo pri razvoju softvera. Preporučujemo čitaocima da pri upoznavanju razvojnih tehnika uvek to imaju u vidu i da pokušaju da u tom kontekstu što bolje sagledaju doprinose svake od tehnika predstavljenih u ovoj knjizi.