SLOBOMIR P UNIVERZITET
FAKULTET ZA INFORMACIONE TEHNOLOGIJE

DIPLOMSKI RAD

PREGLED METODOLOGIJA I TEHNIKA ZA RAZVOJ SOFTVERA ZA GENERISANJE ASP.NET WEB FORMI KAO PRIMJERA DATA-DRIVEN PROGRAMIRANJA

Mentor:
doc. dr Đorđe Babić

Student:
Željko Gavrić
Broj indeksa: 43/08

Doboj, jul 2012.

S A D R Ž A J

1.Uvod

Metodologije razvoja softvera predstavljaju okvir za razvoj samog softvera. Primjena metodološkog razvoja predstvlja ključnu odluku pri izradi svakog projekta. Koju metodologiju koristi? Pitanje izbora metodologija se proteže od njihovog nastankapa do danas. Svaka metodologija povlači za sobom određene prednosti i mane. Veoma je korisno sagledati sve aspekte samog projekta prije izbora metodologije. Glavni apsekti koje treba proučiti su:

- vrijeme koje je na raspolaganju za razvoj,
- da li su zahtjevi precizno definisani prije samog početka rada na projektu,
- da li tokom razvoja može doći do izmjena zatijeva,
- koliki budžet je na raspolaganju,
- kolika je obimnost projekta,
- da li će u budućnosti biti potrebe za proširenjem projekta,
- i drugi.

Potrebno je znati da pri izboru neadekvatne metodologije tok razvoja softvera može otići u potpuno krivom smijeru, te tako ugroziti završavanje projekta.

Programi za automatsko generisanje koda predstavljaju veliko olakšanje svakom programeru, omogućujući mu da veoma brzo dođe do gotovog ili djelimično gotovog softvera. Polazeći od činjenice da web forme čine najveći dio jedne web aplikacije, razvijena je AutoGen aplikacija. To je aplikacija koja automatski generiše ASP.NET web forme na osnovu podataka iz baze podataka, omogućujući funkcije za kasniju manipulaciju tim podacima na web formi. AutoGen aplikacija omogućava korisniku da veoma lako dođe do funkcijalne web forme, ali mu uz to pruža i veliku dozu fleksibilnosti.

2. Metodologije razvoja softvera

Metodologija razvoja softvera ili metodologija razvoja sistema je šablon koji se u softverskom inžinjerstvu koristi za struktuiranje, planiranje i kontrolu procesa razvoja informacionog sistema. Metodologije su se počele koristiti šezdesetih godina prošlog vijeka. Prema većini izvora najstarija primjenjivana metodologija je SDCL (životni ciklus razvoja softvera). Osnovna ideja SDCL-a bila je da se razvoj informacionog sistema obavlja na organizovan i metodičan način, zahtijevajući da softver prođe kroz više faza životnog ciklusa, od nastanka ideje, do isporuke konačnog sistema, u zavisnosti od konkretne metodologije koja se koristi. Glavni cilj metodologije je da razvije funkcionalne poslovne sisteme, koji su laki za upotrebu, ali i dalje nadograđivanje.

Kao imenica, metodologija razvoja softvera je okvir koji se koristi za organizovanje, planiranje i kontrolu procesa razvoja informacionog sistema – uključujući definisanje konkretnih rezultata i artefakata koji su kreirani od strane projektnog tima. Tokom proteklih godina razvio se širok spektar metodologija, od kojih svaka posjeduje određene prednosti i mane. Jedna metodologija nije nužno pogodna za upotrebu od strane bilo kog projektanta. Svaka od metodologija je pogodna za određenu vrstu projekata, na osnovu različitih tehničkih, organizacionih, projektnih i timskih razloga. Okviri za razvoj softvera su često povezani sa nekom vrstom organizacije, koja se razvija, te tokom razvoja koristi i unapređuje metodološki okvir. Okvir metodologije je često opisan u formalnoj dokumentaciji softverskog sistema. Konkretni okviri razvoja softvera su:

- Rational Unified Process (RUP, IBM), od 1998. godine
- Agile Unified Process (AUP), od 2005. godine

Kao glagol, metodologija razvoja softvera je pristup, korišćen od strane organizacija i projektnih timova, koji podrazumijeva primjenu okvira metodologije razvoja softvera (imenica). Konkretne metodologije razvoja softvera (glagol) uključuju:

- Strukturno programiranje od 1969. godine
- Cap Gemini SDM (System Development Methodology), porijeklom iz Pandate, prvi put preveden na engleski jezik 1974. godine
- Strukturno sistemska analiza i projektovanje metoda (SSADM) od 1980. godine, pa nadalje

- Objektno orijentisano programiranje, razvijeno u ranim šezdesetim prošlog vijeka, a postalo dominantan pristup programiranju tokom devedesetih godina prošlog vijeka
- Rapid application development (RAD), od 1991. godine
- Dynamic systems development method (DSDM), od 1994. godine
- Scrum, od 1995. godine
- Timski razvoj sofvera, od 1998. godine
- Ekstremno programiranje, od 1999. godine

Bilo koji od pristupa metodologiji razvoja softvera služi kao osnova za primjenu konkretnog okvira za razvoj i održavanje softvera. Neki od pristupa koji se koriste od nastanka informacionih tehnologija su:

- Waterfall: linearni šablon
- Prototyping: iterativni šablon
- Incremental: kombinovani linearno-iterativni šablon
- Spiral: kombinovani linearno-iterativni šablon
- Rapid application development (RAD): iterativni šablon
- Extreme Programming

2.1. Rapid application development

Rapid application development (RAD) je metoda za razvoj softvera uvedena 1990-ih. U odnosu na tadašnje metode koje ističu pažljivo i dugotrajno prikupljanje zahtjeva prije nego što započnu sa stvarnim razvojem softvera, RAD potiče stvaranje brze prototip verzije softvera koja ispunjava većinu korisnikovih zahtjeva, ali ne nužno sve. Razvoj će se održati u nizu kratkih ciklusa, koji se nazivaju timeboxing, od kojih će se više produbiti funkcionalnosti zahtjeva. Funkcionalnosti koje treba uvoditi u svakom vremenskom okviru su unaprijed dogovorene i plan se strogo poštuje. Naglasak dolazi iz lošeg iskustva s drugim razvojnim metodama u kojima se novi zahtjevi dodaju kako projekat evoluira, a imaju tendenciju da naprave haos i ometaju već pažljivo pripremljene planove i rasporede razvoja. RAD

metodologija se zalaže da razvoj treba činiti od strane malih i iskusnih timova pomoću CASE (Computer Aided Software Engineering) alata kako bi se poboljšala njihova produktivnost.

RAD zagovornici vjeruju da je razvoj brzih prototipa dobar način da se ostvare zahtjevi kupaca dobijanjem neposredne povratne informacije od klijenta. Jedan od problema koji je identifikovan kod drugih metoda razvoja softvera je da klijenti često ne znaju šta stvarno traže, dok ne vide praktičnu implementaciju prototipa. Kao nepoželjan razvoj obično se identifikuje onaj kod koga bi moglo doći do poigravanja sa dogovorenim planovima.

S naglaskom na male timove i kratke razvojne cikluse, ne čudi to da je RAD u doktrini, kod ponovnog korišćenja koda cijenjen kao sredstvo za uspješniji rad. Ovo je izazvalo da RAD kupci rano prigrle objektno-orijentisane jezike i praksu prije nego što su oni stvarno ušli u mainstream (matične tokove). Na slici 1 je prikazan tipičan razvoj RAD metodom.

Slika 1 - Životni ciklus RAD metode

Ključni element RAD-lite pristupa je vizuelno programiranje. Prema ovom konceptu, trebalo bi biti moguće kreirati softver s malo ili bez znanja programiranja. Ideja je da se programi mogu kreirati od ne-programerskih komponenti zajedno, u nekakvoj vrsti radionice, poput razvoja aplikacije. Opet, ta ideja nikad nije u potpunosti ostvarena, ali metoda vizuelnog

razvoja je postala standardni dio tipičnog seta alata programera i sada se rutinski koristi za razvoj nekih dijelova softverskih aplikacija, dok se tradicionalno kodiranje koristi za ostatak. Grafički interfejsi se, na primjer, skoro uvijek kreiraju vizuelno, tako što programeri ili dizajneri korisničkog interfejsa modifikuju željeni izgled korisničkog interfejsa unutar vizuelnog editora i RAD alat zatim automatski generiše odgovarajući kod za kreiranje željenog izgleda. Automatski generisan kod tada čini kostur okruženja aplikacije u cjelini koju softverski inžinjeri izgrađuju i uređuju.

Danas je izraz RAD izgubio većinu svog izvornog značenja, pa čak i u redovima IT profesionalaca. Mnogi nisu svjesni da je RAD referenca formalne metodologije razvoja softvera. Gotovo svaki softverski alat koji se koristi u stvaranju drugog softvera biti će opisan u svojoj marketinškoj literaturi kao nešto što uključuje brzi razvoj aplikacija. Kada se koristi neformalno, izraz brzi razvoj aplikacija obično ukazuje na to da spomenuti alat ima mogućnosti da automatski generiše dio programskog koda. Programski alati koji se danas koriste od strane većine programera, za razvoj novog softvera nazivaju se Integrisana razvojna okruženja (Integrated Development Enviroments - IDEs). Većina njih uključuje neke mogućnosti RAD-a. Prilikom kreiranja novog programa, na primjer, softverski inžinjer može ukazati kakvu će primjenu aplikacija imati i kakva bi trebala biti, konzolna aplikacija, program s grafičkim korisničkim interfejsom ili program koji radi sa bazom podataka. IDE će generisati osnovni šablon koda koji programer uzima kao polaznu tačku za svoj dalji vlastiti rad.

2.2. Model vodopada

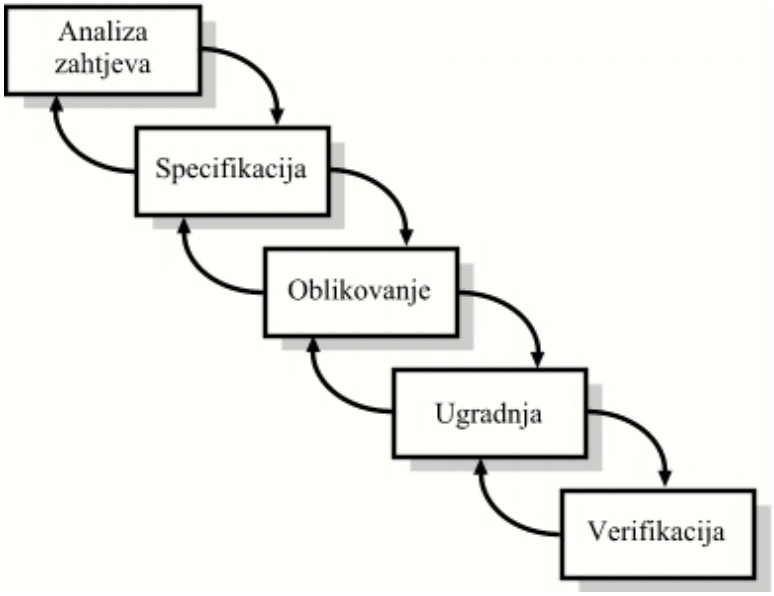
Jedan od najstarijih modela životnog ciklusa softvera je model vodopada. Iako postoji puno problema kod korišćenja modela vodopada, on služi kao osnova za druge, efikasnije modele životnog ciklusa. U modelu vodopada, projekat prolazi kroz faze, od inicijalnog koncepta softvera do testiranja sistema.

Projekat se ispituje na kraju svake faze, da bi se odredilo da li može preći u sljedeću fazu.Ukoliko projekat iz nekog razloga nije spreman za prelazak u sljedeću fazu, on ostaje u tekućoj fazi sve dok ne zadovolji uslove za prelazak u sljedeću fazu.

Model vodopada je baziran na dokumentima, što znači da je dokument osnova koja se prenosi iz jedne faze u drugu.U izvornom modelu vodopada, faze su nepovezane, tj. ne dolazi do preklapanja faza.

Izvorni model vodopada ostvaruje vrlo dobre rezultate za proizvodne cikluse u kojima su definicije proizvoda stabilne. U takvim slučajevima, model vodopada pomaže da se greške otkriju u ranim fazama projekta. On pruža stabilnost zahtjeva, što je veoma značajno za programere. Ukoliko se pravi dobro definisana verzija za održavanje postojećeg proizvoda, ili postojeći proizvod prelazi na novu platformu, model vodopada može da bude pravi izbor za brz razvoj. Na slici 2 prikazan je klasičan model vodopada.

Slika 2 – Model vodopada



Izvor: Harapin, G., "Interakcija čovjek-računalo", http://web.zpr.fer.hr/ergonomija/2000/harapin/index.html, pristupano 21.06.2012.

Pošto se kod modela vodopada planiranje radi unaprijed, smanjuje se opterećenje planiranjem. Rezultati na samom softveru nisu vidljivi do okončanja životnog ciklusa, ali za nekoga ko ima ranijih iskustava sa ovim modelom, moguće je da na osnovu dokumentacije uvidi napredak kroz životni ciklus.

Model vodopada pokazuje dobre rezultate kod velikih, ali jasnih projekata. Prvenstveno dobro radi ukoliko su zahtjevi za kvalitetom projekta postavljeni ispred troškovnih i vremenskih zahtjeva. Ukoliko se eliminišu izmjene tokom implementacije, dolazi do smanjenja potencijalnih grešaka. Osnovna mana izvornog modela vodopada je to što je neophodno da se korisnički zahtjevi specifikuju prije projektantskog rada i prije pisanja programskog koda, a to u većini projekata predstavlja problem. Može se zaključiti da model vodopada nije fleksibilan u pogledu korisničkih zahtjeva.

Neophodno je veoma dobro specifikovati korisničke zahtjeve na početku projekta, jer svaki neprecizno implementiran zahtjev kasnije izaziva velike probleme. Preskočen zahtjev može se uočiti tek u fazi testiranja, što je kasno. Kod ovog modela nije nemoguće vraćanje u prethodnu fazu, ali je sam proces vraćanja veoma komplikovan.

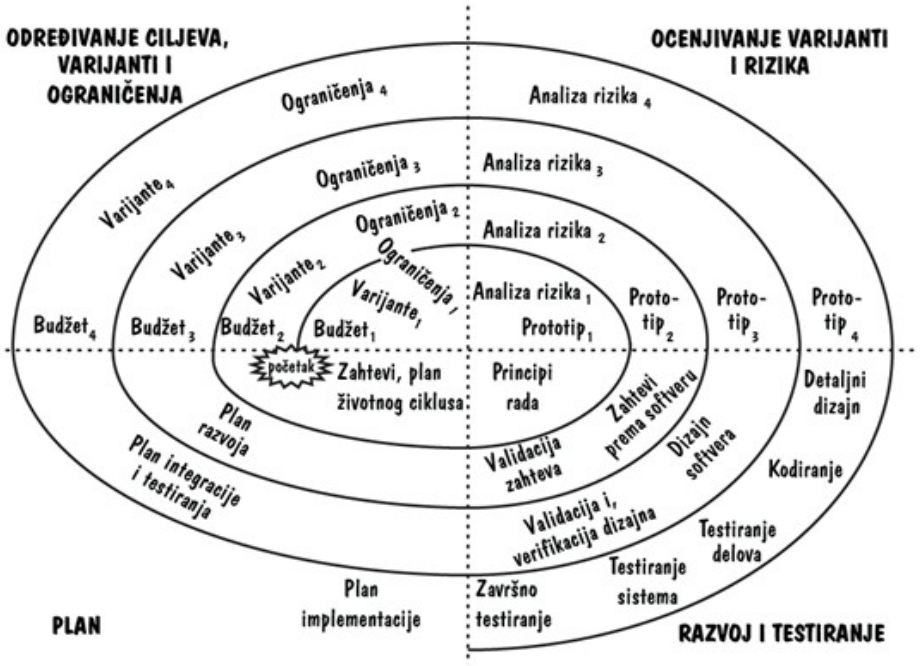
Većina slabosti u ovom modelu nastaje kao posljedica nezavisnog tretiranja aktivnosti, a ne od samih aktivnosti.

Sve druge metodologije predstavljaju varijacije modela vodopada, a razlikuju se uglavnom u brzini implementacije, tipu isporuke i nivou fleksibilnosti.

2.3. Spiralni model

Spiralni model je model razvoja softvera koji razbija projekat u više manjih projekata, a akcenat stavlja na rizik. Svaki mini projekat obrađuje više glavnih rizika ili samo jedan. Koncept rizika može da se odnosi na loše protumačene korisničke zahtjeve, lošu arhitekturu, potencijalne probleme sa performansama, probleme u osnovnoj tehnologiji, itd. Na slici 4 prikazan je spiralni model.

Slika 3 – Spiralni model



Izvor: Elektronsko učenje, http://link-elearning.com, pristupano 21.06.2012.

Osnovna ideja dijagrama spiralnog modela je da se počne sa malim koracima u sredini, da se ispitaju rizici, da se napravi plan za obradu rizika, i onda se posveti pristupu za sljedeću iteraciju.

Svaka iteracija uključuje šest koraka:

1. Određivanje ciljeva, alternativa i ograničenja
2. Identifikovanje i otklanjanje rizika
3. Ocjena alternativa
4. Razvoj isporuka za datu iteraciju i provjera da li su ispravne
5. Plan sljedeće iteracije
6. Posvećenje pristupu za sljedeću iteraciju (ukoliko postoji)

Ovaj model je moguće kombinovati sa drugim modelima životnog ciklusa softvera. Kombinovanje se može raditi na nekoliko različitih načina. Projekat je moguće započeti iteracijama za smanjenje rizika, a zatim nastaviti sa nekim drugim modelom koji nije baziran na riziku. Moguće je ugraditi neke druge modele u spiralni model.

Glavna prednost spiralnog modela je da se sa povećanjem troškova smanjuje rizik. Na kraju svake iteracije postoje kontrolne tačke. Ukoliko projekat nije moguće završiti unutar zadatih rokova zbog tehničkih ili nekih drugih razloga, spiralni model omogućava to saznanje dosta rano. Samim tim što je moguće rano otkrivanje probijanja rokova, cijena produženja roka ne košta puno. Jedna od prednosti spiralnog modela je hvatanje u koštac sa neizbježnim promjenama koje razvoj softvera nameće.

Osnovna mana spiralnog modela je njegova složenost. Spiralni model zahtijeva da upravljanje projektom bude savjesno i brižno. Nedostatak spiralnog modela je što su procjene koje se tiču koštanja projekta neprecizne. Osnovni razlog grubih procjena je što određene analize nisu završene sve dok te etape ne prođu fazu projektovanja.

Spiralni razvoj se ne koristi u velikoj mjeri, ali je ostavio trag na modernije metode koje se danas koriste, prvenstveno na agilne metode.

2.4. Rational jedinstveni proces razvoja

Rational jedinstveni proces razvoja softvera (Rational Unified Process) je okvir za iterativni razvoj softvera koji je kreiran od strane kompanije Rational Software. RUP nije

konkretan perspektivan proces, ali je prilično prilagodljiv okvir. RUP obično biraju kompanije ili timovi koji koriste jedan njegov dio koji odgovara njihovim potrebama. On predstavlja specifičnu implementaciju jedinstvenog razvojnog procesa.

RUP je proizvod softverskog procesa, koji je originalno razvijen od strane kompanije Rational Software, koju je u februaru 2003. godine kupila firma IBM. On uključuje bazu znanja sa primjerima artefakata i detaljnim opisima za mnoge različite vrste aktivnosti. RUP je uključen u IBM Rational Method Composer, proizvod koji omogućava prilagođavanje procesa. Kombinovanje iskustava preduzeća dovelo je do artikulacije šest najboljih praksi modernog softverskog inžinjerstva:

1. Iterativni razvoj
2. Upravljanje zahtjevima
3. Korišćenje komponenata na bazi strukture
4. Vizuelno modelovanje softvera
5. Kontinualna provjera kvaliteta
6. Kontrola promjena

Prethodno nabrojane prakse su dovele do usavršavanja Rational procesa razvoja. RUP se najviše koristi od strane terenskih timova da poboljša kvalitet i predvidljivost u njihovim naporima da razviju šo bolji softverski proizvod. RUP je kompletirao strateški okvir za Rational:

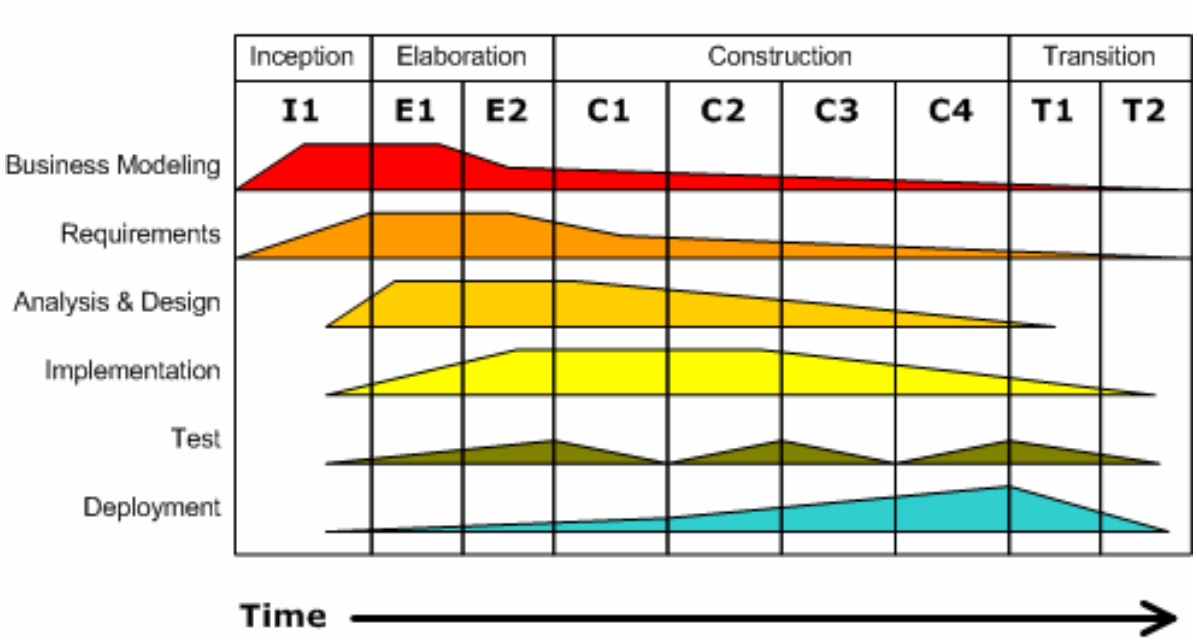
- Prilagođavanje procesa koji vodi razvoj
- Alati koji automatizuju taj proces
- Servisi koji ubrzano usvajaju oboje, te proces i alati

Zasnovan na skupu gradivnih blokova koji opisuju ono što rade, vještine koje su neophodne i objašnjenja kako postići specifične razvojne ciljeve. Osnovni gradivni blokovi RUP-a su sljedeći:

- Uloge (ko) – Uloga definiše skup srodnih vještina, kompetencija i odgovornosti.
- Radni proizvodi (šta) – Radni proizvod predstavlja nešto što proizilazi iz zadatka, uključujući sve dokumente i modele koji su nastali tokom izvršavanja procesa.
- Zadaci (kako) – Zadatak opisuje jedinicu rada koja je dodijeljena ulozi koja obezbjeđuje značajan rezultat.

Na slici 4 prikazane su faze i discipline Rational jedinstvenog razvojnog procesa.

Slika 4 – Faze i discipline RUP-a



Izvor: Wikipedia, "Rational Unified Process", http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process, pristupano 22.06.2012.

Unutar svake iteracije zadaci su razvrstani u devet disciplina:

- Šest inžinjering disciplina
  - Poslovno modelovanje
  - Zahtjevi
  - Analiza i dizajn
  - Implementacija
  - Testiranje
  - Razvoj
- Tri prateće discipline
  - Podešavanje i upravljanje promjenama
  - Upravljanje projektima
  - Okolina

Životni ciklus razvoja softvera pomoću RUP-a se sastoji od četiri faze. Ove faze omogućavaju proces koji će biti predstavljen na visokom nivou, na sličan način kao kod

modela vodopada. U suštini, ključ leži u iteraciji koja se proteže kroz sve faze. Virtualizacija faza i disciplina tokom vremena se naziva uzvišenje na grafikonu.

Faze životnog ciklusa su:

- Početna faza:
  - Vizija, zahtjevi visokog nivoa, poslovni slučajevi korišćenja
  - Identifikovanje rizika
  - Procjena troškova, vremena, plana i kvaliteta proizvoda
  - Inicira se kreiranje poslovne studije opravdanosti ulaženja u projekat
- Faza elaboracije:
  - Osnovna arhitektura, većina zahtjeva je detaljna
  - Dizajn nije detaljan
  - Precizna procjena resursa i vremena
- Faza konstrukcije:
  - Rad na proizvodu, završno testiranje sistema
- Faza prelaza:
  - Prihvatljivost od strane stejkoldera

2.5. Larmanova metoda

Larmanova metoda bazirana je na iterativno – inkrementalnom principu. Koristi upravljanje prema slučajevima korišćenja, dok se projektovanje odvija prema objektnim principima. Notacija se predstavlja preko UML dijagrama.

Razvoj softvera kod Larmanove metode se odvija u 5 faza, a to su:

1. Opis zahtjeva i slučajevi korišćenja
2. Analiza
3. Projektovanje
4. Implementacija
5. Testiranje

U prvoj fazi se definišu zahtjevi koje softver treba da ispuni. Zahtjevi se definišu pomoću modela slučaja korišćenja, kojim se opisuje skup željenih korišćenja sofvera od strane

korisnika. U ovoj fazi se identifikuju učesnici u sistemu i slučajevi korišćenja softvera. Prvo se korisnički zahtjevi opisuju tekstualno, pa se zatim prelazi na kreiranje slučajeva korišćenja, koji se opisuju tekstualno (obično u vidu određenog šablona), a zatim i grafički. Primjer tekstualnog opisa slučaja korišćenja dat je u nastavku.

SK1: Slučaj koriščenja – Logovanje

Naziv SK
Logovanje

Aktori SK
Operater

Učesnici SK
Operater i sistem (program)

Preduslov: Sistem je uključen. Sistem prikazuje formu za logovanje.

Osnovni scenario SK

1. Korisnik **unosi** podatke o korisniku.(APUSO)
2. Korisnik **provjerava** da li je uneo sve potrebne podatke.(ANSO)
3. Korisnik **poziva** sistem da izvrši proveru.(APSO)
4. Sistem **pretrazuje korisnike**.(SO)
5. Sistem **prikazuje** korisniku početnu formu.(IA)

Alternativna scenarija
5.1 Ukoliko sistem ne može da pronađe operatera koji se loguje prikazuje korisniku poruku "Sistem ne može da nađe korisnika sa unesenim vrijednostima. Slog ne postoji u bazi podataka". Prekida se izvršenje scenarija. (IA)

Faza analize opisuje logičku strukturu i ponašanje softverskog sistema, tj. poslovnu logiku softverskog sistema.[1]

Ponašanje se opisuje pomoću:

- sistemskih dijagrama sekvenci, koji se prave za svaki prethodno utvrđen SK
- ugovora o sistemskim operacijama, koje se dobijaju na osnovu sistemskih dijagrama sekvenci.

Struktura se opisuje pomoću:

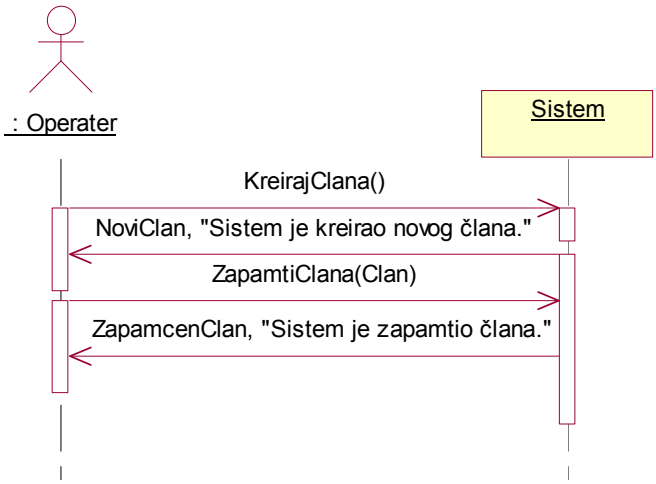
- konceptualnog modela i

[1] Vlajić, S. (2011), Projektovanje softvera (Skripta), FON, Beograd, str. 15.

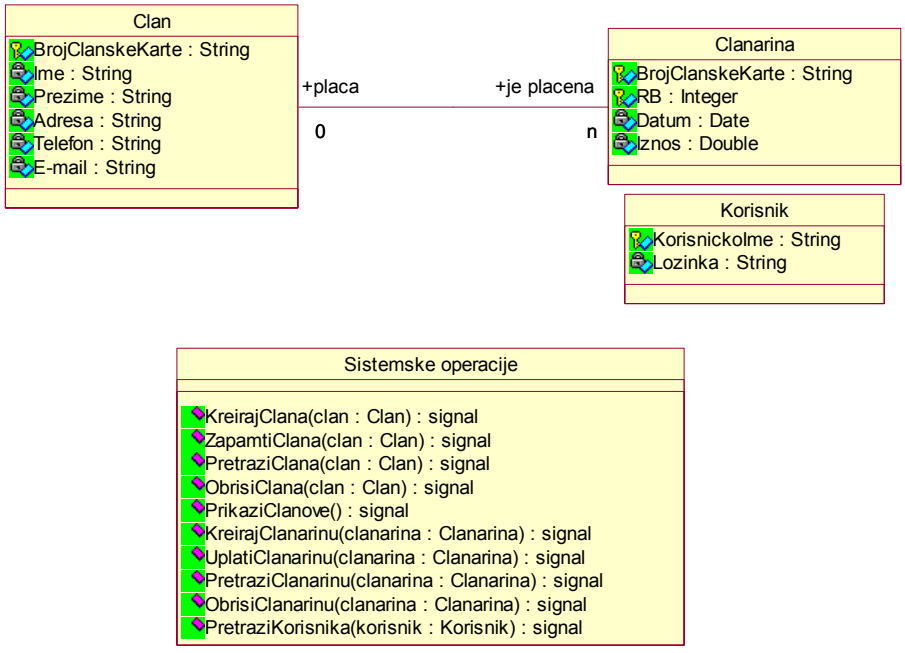
- relacionog modela.

Na slici 5 prikazan je primjer sistemskog dijagrama sekvence, a na slici 6 primjer konceptualnog modela konkretnog sistema.

Slika 5 – Sistemski dijagram sekvence

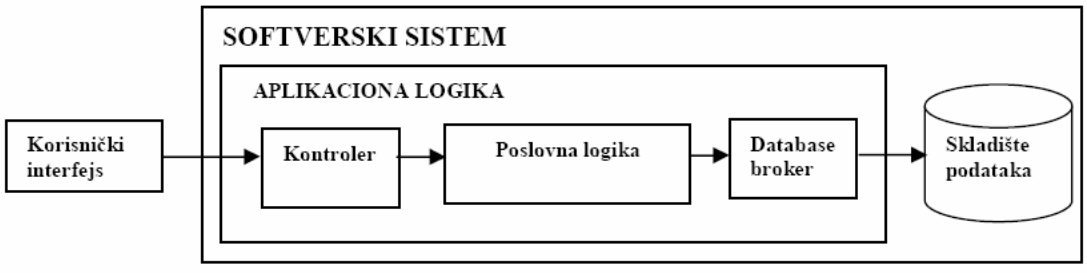


Slika 6 – Konceptualni model



Faza projektovanja opisuje fizičku strukturu i ponašanje softverskog sistema.[2] Projektovanje arhitekture softverskog sistema obuhvata projektovanje aplikacione logike, skladišta podataka i korisničkog interfejsa. U okviru aplikacione logike projektuju se kontroler, poslovna logika i broker baze podataka. Projektovanje poslovne logike obuhvata projektovanje logičke strukture i ponašanje softverskog sistema. Na slici 7 prikazan je izgled softverskog sistema koji zadovoljava tronivojsku arhitekturu.

Slika 7 – Izgled softverskog sistema



Izvor: Vlajić, S. (2011), Projektovanje softvera (Skripta), FON, Beograd, str. 28.

Nakon faze projektovanja može se preći na fazu implementacije. Implementacija se radi u nekom od programskih alata, pisanjem koda za klase koje su u ranijim fazama precizno i tačno definisane.

Testiranje, shodno arhitekturi softverskog sistema, može da se podijeli u nekoliko nezavisnih jedinica testiranja. Nezavisne jedinice testiranja su softverske komponente koje su dobijene u fazi implementacije softverskog sistema.

Testiranje svake softverske komponente podrazumijeva pravljenje:

- Test slučajeva koji opisuju šta test treba da provjeri
- Test procedura koje opisuju kako će se izvršiti test
- Test komponente koje treba da automatizuju test procedure ukoliko je to moguće

Nakon testiranja softverskih komponenti vrši se njihova integracija. U tom smislu se prave testovi integracije softverskih komponenti. Svaki slučaj korišćenja je testiran, pri tome se u procesu testiranja forme popunjavaju ispravnim, kao i neispravnim podacima, čiji je cilj bio da se vidi reagovanje sistema na sve moguće situacije koje mogu da se pojave.

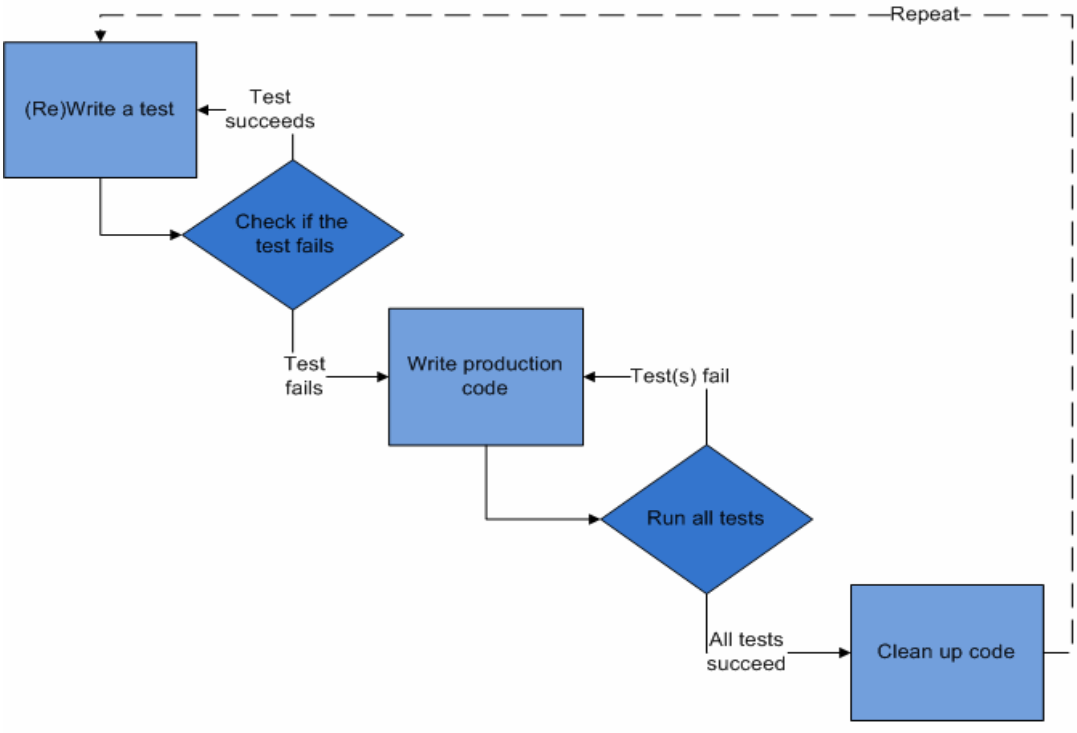
[2] Vlajić, S. (2011), Projektovanje softvera (Skripta), FON, Beograd, str. 27.

2.6. Razvoj vođen testovima

Razvoj vođen testovima (Test-driven development) je tehnika razvoja softvera koja se oslanja na ponavljanje vrlo kratkih razvojnih ciklusa. Sama tehnika se zasniva na stalnom testiranju napisanog programskog koda. Programer prvo piše automatizovani test, a nakon toga piše kôd koji treba da zadovolji taj test. Nakon toga se taj kôd refaktoriše kako bi zadovoljio zahtijevane standarde. Kada se dese izmjene u nekom od testova neophodno je da se taj test ponovo verifikuje bez obzira na to da li je ranije bio verifikovan. Pored ovakvih, jediničnih testova, postoje i testovi koji testiraju funkcionalne cjeline. Ovi testovi se nazivaju test slučajevi i obično ih izvodi sam klijent. TDD obezbjeđuje brzu povratnu vezu (feedback).

Neophodno je da programeri pravilno napišu testove koji neće proći određen dio koda da bi uvidjeli kako je riješen slučaj kada program treba da uhvati grešku, tj. da bi se uvjerili da neće doći do iznenadnog prekida rada programa. Na slici 8 prikazan je razvojni ciklus TDD-a.

Slika 8 – Razvojni ciklus TDD-a



Izvor: Wikipedia, " Test-driven development", http://en.wikipedia.org/wiki/Test-driven_development, pristupano 22.06.2012.

Razvojni ciklus se satoji iz sledećih faza:

- Dodavanje – pisanje testa
- Pokretanje svih testova i uočavanje testa koji ne prolazi
- Pisanje koda za test koji ne prolazi
- Pokretanje automatizovanih testova i provjeravanje da li su uspješni
- Refaktorisanje i čišćenje koda
- Ponavljanje

U razvoju vođenom testovima, dodavanje nove funkcije počinje pisanjem testa. Ovaj test mora biti takav da ne prođe, jer se on piše prije implementiranja funkcije. Ukoliko test prođe znači da funkcija postoji ili test nije napisan korektno. Da bi programer napisao test mora savršeno dobro poznavati specifikaciju funkcije za koju piše test i sve zahtjeve koje ta funkcija treba da zadovolji. Programer može doći do tih saznanja kroz niz slučajeva korišćenja ili usmenih korisničkih zahtjeva. To može dovesti do varijacije ili modifikacije nekog od postojećih testova. Pisanje testova prije pisanja koda je osnovna razlika između razvoja vođenog testovima i drugih metoda kod kojih se jedinični testovi pišu nakon pisanja koda.

Druga faza potvrđuje da li test radi korektno, tj. da li postoji potreba za pisanjem novog koda. U ovoj fazi testira se sam napisani test. Ukoliko je test negativan, tj. ukoliko ne zadovolji očekivano test se odbacuje, a ukoliko je test pozitivan on se uvažava. Novi test treba da „padne“ očekivanog razloga.

Treća faza je pisanje koda koji će zadovoljiti test koji nije prošao. Novi kôd, koji je napisan u ovoj fazi, obično nije savršen. Ovo je prihvatljivo iz razloga što se test u narednim koracima refaktoriše i prečišćava. U ovom koraku je najvažnije da je kôd napisan i dizajniran tako da zadovolji test.

Ukoliko svi test slučajevi prođu, programer može biti siguran da testirani kôd ispunjava sve postavljene zahtjeve. Ovo je tačka iz koje počinje poslednji korak u razvojnom ciklusu.

Peta faza podrazumijeva čišćenje koda, ukoliko je to potrebno. Ponovnim pokretanjem test slučajeva programer se može uvjeriti da čišćenje i refaktorisanje nije oštetilo neku od postojećih funkcionalnosti. Koncept uklanjanja duplikata, koji je sastavni dio ove faze, je važan aspekt kod razvoja softvera. Ovaj koncept se odnosi kako na duplikate u test kodu, tako

i na duplikate u programskom kodu, ali i na duplikate koji se pojavljuju između test koda i programskog koda.

Počevši sa sljedećim, novim, testom ciklus se ponavlja te se povećava fukcionalnost. Veličina koraka uvijek treba da bude mala. Ukoliko novi kôd ne zadovolji dovoljno brzo novi test, ili drugi testovi neočekivano ne prođu, programer treba da se vrati na prethodni korak. Uzastopna integracija pomaže pružajući revertibilne kontrolne tačke. Ukoliko se koriste eksterne biblioteke važno je da inkrementi ne budu previše mali, jer bi onda mogli efektivno testirati samo biblioteku.

Razvoj vođen testovima je teško koristiti u situaciji kada se zahtjevaju potpuno funkcionalni testovi. Primjeri ovakvih situacija su korisnički interfejsi, programi koji rade sa bazama podataka, kao i neki programi koji zavise od specifičnih mrežnih podešavanja.

2.7. Agilni razvoj softvera

Agilni razvoj softvera predstavlja skup metoda za razvoj softvera zasnovan na iterativnom i inkrementalnom razvoju. Promoviše adaptivno planiranje, evolutivni razvoj i isporuku, vremenski podijeljen iterativni pristup, i ohrabruje brzo i fleksibilno reagovanje na promjene.

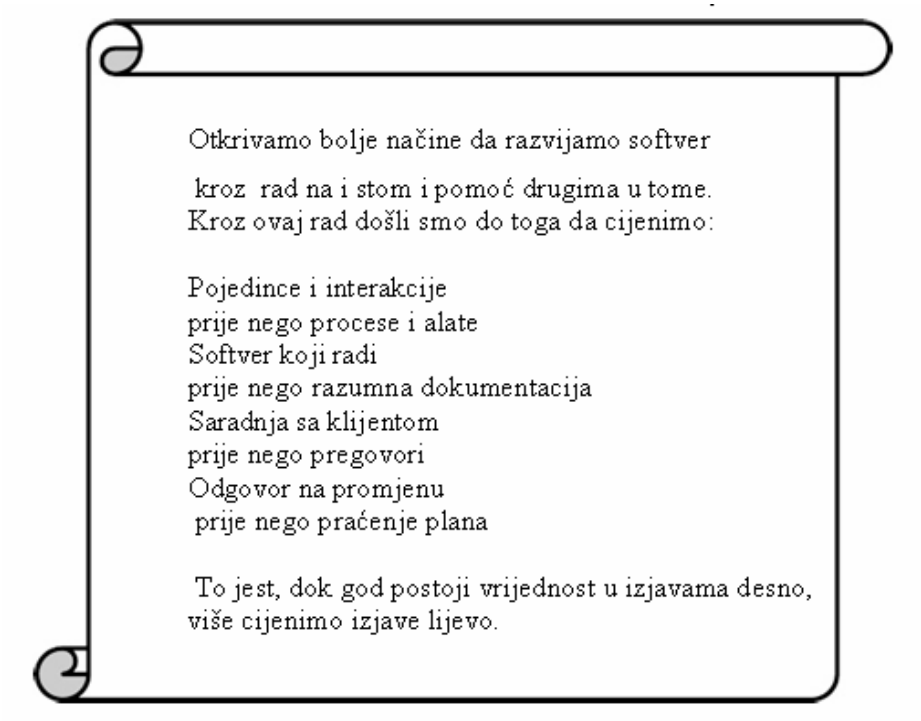
Agilne metode su u početku zvane lagane (lightweight). Godine 2001 istaknuti svjetski softverski inženjeri su se sastali u mjestu Snowbird, Utah, SAD, gde su usvojili naziv “agilne metode.” Neki od njih su kasnije oformili neprofitnu organizaciju “The Agile Aliance” čiji je cilj promovisanje agilnog razvoja. Stvaraoci agilne metodologije kreiraju manifest agilnih metodologija koji propisuje vrijednosti i principe koje moraju prihvatiti sve druge metode da bi bile agilne.

Nastanak ranih agilnih metoda vezuje se za period prije 2000.godine:

- 1986. – SCRUM u oblasti opšteg menadžmenta
- 1995. – Adaptivni razvoj softvera, Razvoj vođen karakteristikama i Metod dinamičnog razvoja sistema
- 1996. – Crystal clear i ekstremno programiranje

Na slici 9 prikazan je manifest agilnih metodologija.

Slika 9 – Manifest agilnih metodologija



Tek sa ekstremnim programiranjem nastaje svijest o novoj vrsti metodologija. Najpoznatije metode agilnog softverskog razvoja danas su:

- Ekstremno programiranje (XP) i Industrijsko ekstremno programiranje (IXP)
- Scrum
- Agilno modeliranje
- Adaptivni razvoj softvera (ASD)
- Crystal clear i ostale crystal metode
- Metod dinamičnog razvoja sistema (DSDM)
- Razvoj vođen karakteristikama (FDD)
- "Suvi" razvoj (lean development)
- Agile Unified Process (AUP)

Agilne metode imaju značajne razlike u odnosu na ranije plansko-centrične inžinjerske metode. Najuočljivija razlika je insistiranje na manje obimnoj dokumentaciji za dati problem. Umjesto dokumentaciono-orjentisane, agilne metode su prije orjentisane ka izvornom kodu kao ključnom dijelu dokumentacije.

Ovo je ipak samo površno gledište. To je samo simptom mnogo dubljih razlika a one su po Martinu Fowleru sljedeće:

- Agilne metode su prije adaptivne nego predvidive.
- Agilne metode su orjentisane ka ljudima radije nego ka procesima.

Kada je potrebno mijenjati projekat i adaptivni tim se takođe mijenja sa njim. Adaptivni tim ima problem da opiše šta će se tačno dešavati u budućnosti. Što je datum o kome se govori dalji, to će tim koji primjenjuje neki adaptivni metod biti nejasniji o detaljima koji bi trebali objasniti šta će se tada dešavati u projektu. Adaptivni tim će tačno opisati šta će se raditi sljedeće nedelje, ali o sljedećem mesecu će imati samo predstavu koje bi osobine trebalo tada dodavati u program. Izveštaji adaptivnog tima o sljedećeih šest meseci mogu sadržati samo informacije o očekivanim rezultatima u zavisnosti od uslova i zahtjeva, kao i zvanične izjave vezane za plan puštanja sljedeće verzije.

Nasuprot ovome predvidive metode se koncentrišu na detaljno planiranje budućnosti. Ovakav tim tačno zna koje osobine i zadaci su planirani u kom trenutku tokom trajanja procesa realizacije softverskog projekta. Međutim, za njega je teško da mijenja pravac djelovanja. Plan je, uglavnom, optimizovan za glavni cilj te promjena pravca može prouzrokovati kompletno odbacivanje dosadašnjeg rada i ponovni početak na novi način. Prediktivni timovi često postavljaju posebnu komisiju za kontrolu promjena kako bi se razmatrale samo one najbitnije.

2.8. Scrum

Scrum predstavlja iterativnu i inkrementalnu metodu razvoja softvera. Inkrementalni razvoj predstavlja razvoj softvera korak po korak, dok iterativni način predstavlja strategiju vremenskog planiranja u kojem se softver kroz svaki definisani period vremena dodatno usavršava. Koristi agilni pristup za upravljanje softverskim projektima i proizvodima. Scrum se metoda koja se pojavila početkom devedesetih godina prošlog vijeka. Ova metoda je više vezana za agilno upravljanje softverskim projektom, nego za agilno projektovanje softvera. Ona propisuje načine upravljanja zahtjevima, formiranje iteracija (planiranje sprinta), kontrole implementacije i isporuke klijentu. Često se upotrebljava kao način vođenja XP-a, ili drugih

projekata koji ne moraju obavezno da se projektuju nekom agilnom metodom. Na slici 10 prikazan je scrum proces.

Osnovu predstavljaju tri ključna pitanja koja se postavljaju na svakodnevnim, jutarnjim "stojećim" petnaestominutnim sastancima, a to su:

1. Šta je urađeno juče?
2. Šta će se raditi danas?
3. Kakve nas danas prepreke očekuju?

Ova pitanja se odnose na:

1. Kontrolu izvršenog
2. Planiranje budućeg dizajna
3. Identifikaciju rizika i nalaženje rešenja

Slika 10 – Scrum proces

Izvor: Murphy, C., "Adaptive Project Management Using Scrum", http://www.methodsandtools.com/archive/archive.php?id=18, pristupano 23.06.2012.

Osnovna vremenska cjelina Scruma jest sprint. Sprint je zaokružena jedinica razvojnog procesa koja najčešće traje 30 dana. Unutar svakog sprinta, Scrum prolazi kroz sve faze razvojnog procesa. Planiranje, programiranje, testiranje i isporuka ponavljaju se kroz svaki sljedeći sprint. Na kraju svakog sprinta razvojni tim isporučuje zaokruženi dio proizvoda, odnosno potencijalno isporučiv inkrement proizvoda. Razbijanjem razvojnog procesa na

sprintove Scrum smanjuje rizik od isporuke lošeg softvera, tj. softvera koji neće zadovoljiti tržište. Softver se zbog toga lakše prilagođava novim i promijenjenim zahtjevima korisnika koji su neizbježni. Ipak, Scrum se ne zaustavlja na tome, pa se svaki pojedini sprint dodatno razbija na još manje dijelove koji traju 24 sata, a počinju i završavaju dnevnim Scrumom.

Dnevni Scrum jedna je on malobrojnih obaveza cijelog razvojnog tima, a nužan je da bi se postigla unutrašnja transparentnost u timu. To je stand-up sastanak na kojem učestvuju svi članovi razvojnog tima, a traje oko 15-ak minuta.

Za provođenje dnevnog Scruma kao i cijelog razvojnog procesa odgovoran je Scrum Master. Scrum Master donekle odgovara ulozi vođe projekta, iako Scrum strogo gledajući ne poznaje ulogu vođe projekta. Scrumov način razvoja softvera u suštini je samoorganizujući proces te kao takav ne zahtjeva ulogu vođe, barem ne u smislu kako se tradicionalno takva uloga opisuje. Scrum Master ima ulogu prvenstveno kontrolisati proces, ali ne i ljude u timu. Umjesto da raspoređuje posao i govori svim članovima tima šta da rade, Scrum Master nadzire proces i brine se o tome da se on poštuje.

2.9. Ekstremno programiranje

Ekstremno programiranje (XP) je metodologija razvoja softvera koja ima za cilj da poboljša kvalitet softvera i reagovanja na promjenljive zahtjeve klijenata. Kao tip agilnog razvoja softvera XP zahtjeva česta izdanja u kratkim razvojnim ciklusima (timeboxing), koja su namijenjena da unaprijede produktivnost i uvedu kontrolne tačke, na mjestima gdje bi novi zahtjevi korisnika mogli biti usvojeni. Iako ekstremno programiranje nije prva metoda agilnog razvoja softvera, ono popularizuje i omasovljava upotrebu agilnih metoda.

Do danas, sve veći broj softverskih kompanija je prešlo na ekstremno programiranje. Vid njihovog organizovanja može biti drugačiji u svakom konkretnom slučaju i za svaki pojedinačni zahtjev, jer ekstremno programiranje ne predstavlja krutu metodologiju ili metodu, već je to jezik uzora koji dozvoljava taktičke izmjene u proceduri i sekvenci izvođenja pojedinih aktivnosti. Zajedničko za sve je korišćenje obaveznih praktičnih mehanizama. Čak i sprovođenje tih praksi podliježe prilagođavanju, zavisno od potrebe, što i jeste osnovna odlika uzora kao novog načina naučnog sagledavanja pojava koje se proučavaju.

Definicija ekstremnog programiranja opisuje ekstremno programiranje kao disciplinu za razvoj softvera, koja organizuje ljude da proizvedu softver visoke kvalitete. XP pokušava

da smanji troškove koji nastaju zbog promjena zahtjeva kroz više kratkih razvojnih ciklusa. U ovoj doktrini, promjene su prirodan, neizbježan i poželjan aspekt softverskog razvoja projekta, te ih je zato neophodno planirati. Ekstremno programiranje uvodi nekoliko osnovnih vrijednosti, principa i praksi u okvir agilnog programiranja.

Ekstremno programiranje opisuje četiri osnovne aktivnosti koje se obavljaju u procesu razvoja softvera:

- kodiranje
- testiranje
- slušanje
- projektovanje

Zagovornici ekstremnog programiranja tvrde da je programski kôd jedini važan proizvod procesa razvoja sistema. Pod pojmom programskog kôda podrazumijevaju se softverske instrukcije koje računar može da izvrši. Kodiranje se koristiti da bi se shvatilo koje rješenje je najpogodnije. Takođe se može koristiti da pomogne u razmjeni mišljenja o problemima tokom programiranja. Programer koji se bavi programiranjem kompleksnog problema, mora naći način da predstavi rješenje svojim kolegama, koji će možda predložiti dosta jednostavnije rješenje istog problema. Kako kažu zagovornici ovog modela, kôd uvijek mora biti jasan i precizan i ne može se tumačiti na više načina.

Pristup ekstremnom programiranju kaže da malo testiranja može da eliminiše nekoliko mana, dok dosta testiranja može eliminisati mnogo više mana. Pojedinačni testovi testiraju određenu funkcionalnost i treba da pokažu da li data funkcionalnost radi kako bi trebala. Programer piše velik broj automatizovanih testova na mjestima gdje očekuje da bi moglo doći do greške, a ako svi od tih testova prikažu očekivani rezultat, kodiranje je završeno. Prije početka kodiranja naredne funkcionalne cjeline, neophodno je testirati svaki dio koda koji je napisan u tekućoj cjelini. Testovi prihvatljivosti potvrđuju da zahtjevi, koje je shvatio programer, zadovoljavaju stvarne potrebe naručioca.

Programeri moraju saslušati korisnikove zahtjeve u pogledu toga šta sistem treba da radi, tj. da moraju shvatiti poslovnu logiku sistema. Oni to moraju razumjeti dovoljno dobro da bi korisniku mogli dati informacije o tome kako se problem može ili ne može riješiti, te kako bi mu predložili eventualno alternativno rješenje. Komunikacija između programera i klijenta je najneophodnija u fazi planiranja.

Sa stanovišta jednostavnosti, moglo bi se reći da sistem ne treba više kodiranja, testiranja i slušanja, ukoliko se ove aktivnosti prvobitno dobro obave. U praksi to nije moguće. Moguće je proći dug put bez projektovanja, ali kako sistem postaje kompleksniji povećava se vjerovatnoća da će doći do zastoja. Određeni problemi se mogu izbjeći kreiranjem strukture projektovanja koja prikazuje logiku sistema. Dobra struktura će pomoći u izbjegavanju zavisnosti u sistemu, što znači da izmjena u jednom dijelu sistema ne utiče na drugi dio sistema.

Glavni problemi u timu često potiču od nedostatka komunikacije. Na taj način se sprečava protok informacija i podstiče softverska entropija, koja je glavni neprijatelj uspjehu projekta. Razvoj komunikacije među članovima tima, komunikacije sa rukovodstvom, te razvoj komunikacije sa klijentima, što je najbitnije, treba da bude najprioritetniji zadatak svakoga u timu.

Posvećenost jednostavnosti je definitivno jedna kontraverzna vrijednost koja je od velikog značaja za XP tim. Bez jednostavnosti nema evolutivnog pristupa dizajnu, niti brzog odziva na promjene.

Povratna sprega je važna, jer omogućava povratnu informaciju, kao indikator za evaluaciju. Povratna sprega sa klijentima, drugim programerima, konkretnim kodom i dosadašnjim sistemom čine vrijednosti koje se moraju njegovati.

Bez hrabrosti nema uspješnog rada. To je hrabrost za prihvatanje promjena, hrabrost za promovisanje ideja, kao i hrabrost za odbacivanjem dijelova koda i ponovnim početkom, ako za to bude potrebe.

Iz svih ovih vrijednosti proizilazi poštovanje. Ono se odnosi na poštovanje kolega u timu. Onaj ko vodi računa o prethodnim vrijednostima, sigurno dobija poštovanje ostalih. U XP timu niko ne smije biti zapostavljen, ili odbačen, prije svega zato što to vodi lošoj atmosferi u timu i izaziva pad zainteresovanosti za realizacijom projekta. Na taj način se projekat ugrožava i u najgorem slučaju, napušta. Postoji dvanaest praktičnih mehanizama kojih se treba pridržavati pri razvoju softvera pomoću XP. To su:

- Programiranje u paru
- Paniranje igre
- Razvoj vođen testovima (TDD)
- Cjelokupnost tima
- Stalna integracija

- Poboljšanje dizajna (refaktorisanje)
- Male verzije
- Standardi kodiranja
- Kolektivno vlasništvo koda
- Jednostavan dizajn
- Metafora u sistemu
- Održivi korak

Svi ovi praktični mehanizmi moraju biti motivisani i vođeni vrijednostima.

2.10. Lean razvoj softvera

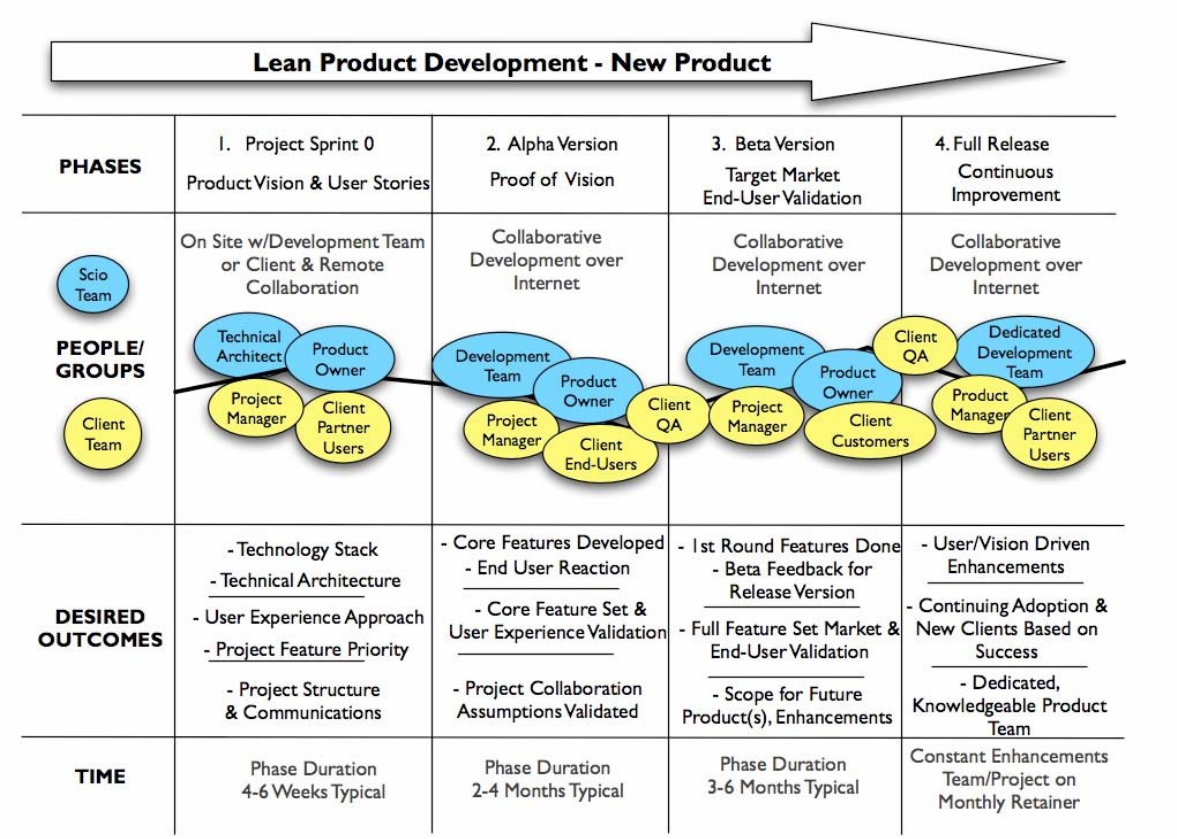
Pojam Lean potiče iz istoimene knjige koju su napisali Mary Poppendieck i Tom Poppendieck. Knjiga predstavlja tradicionalne Lean principe u izmijenjenom obliku, kao skup od 22 alata i poredi te alate sa agilnim principima. Angažman Poppendieck-a u zajednici agilnog razvoja softvera, ukjlučujući i razgovore na nekoliko konferencija o agilnom razvoju softvera rezultovao je konceptima koji su prihvaćeni od strane Agile zajednice.

Lean razvoj se može sažeti u sedam faza, koje su konceptualno veoma slične Lean proizvodnim pricipima:

- Eliminacija nepotrebnog
- Pojačano učenje
- Odlučivanje u poslednjem momentu
- Dostavljanje u najkraćem mogućem roku
- Osnaživanje tima
- Kreiranje integriteta
- Vidjeti cjelinu

Na slici 11 prikazan je tok Lean razvoja softvera.

Slika 11 – Lean razvoj softvera



Izvor: Dunham, M., " Lean Software Product Development in 4 Phases", http://blog.sciodev.com/2010/02/24/lean-software-product-development-in-4-phases, pristupano 23.06.2012.

Nepotrebnim se smatra sve što ne predstavlja povećanje vrijednosti za kupca. Ova definicija uključuje:

- nepotreban kôd i funkcionalnost
- kašnjenje u procesu razvoja softvera
- nejasni zahtjevi
- nedovoljno testiranje
- birokratija
- spora interna komunikacija

Da bi se određeni dijelovi mogli eliminisati, neophodno je identifikovati šta je to nepotrebno. Ukoliko se neka aktivnost može zaobići, znači da je ona višak. Procesi i funkcije koje se ne koriste od strane kupca takođe predstavljaju višak koji treba eliminisati.

Razvoj softvera predstavlja kontinuirani poces učenja. Najbolji pristup za poboljšanje okruženja za razvoj softvera je da se pojača učenje. Akumulaciju grešaka treba spriječiti testiranjem, odmah nakon što je kôd napisan. Umjesto pisanja prekomjerne dokumentacije ili detaljnog planiranja, ideje treba usmjeriti na pisanje koda i izgradnju. Proces prikupljanja korisničkih zahtjeva može biti pojednostavljen pokazujući im određene primjere i dobijanjem povratnih informacija od njih.

Razvoj softvera uvijek je povezan sa određenom dozom nesigurnosti. Bolje rezultate moguće je postići primjenom opcija – zasnovanog pristupa, odlaganjem odluka koliko god je to moguće, do trenutka do kad sistem može biti zasnovan na činjenicama, a ne na nesigurnim pretpostavkama i predviđanjima. Što je sistem složeniji to u njega treba ugraditi više sposobnosti za promjene, što omogućava odlaganje važnih i ključnih odluka.

U eri brze tehnološke revolucije, ne preživljava najveći, nego najbrži. Neophodno je isporučiti krajnji proizvod što ranije, da bi se što prije dobila povratna informacija od korisnika, te ugradila u sljedeću iteraciju. Kraće iteracije poboljšavaju učenje i komunikaciju u timu. Nemoguće je odlagati odluke bez brzine. Brzina zadovoljava sadašnje potrebe kupaca, a ne ono što su oni tražili juče. Kupci mnogo cijene brzu isporuku kvalitetnog proizvoda.

U tradicionalnim sistemima menadžeri govore radnicima kako da rade svoj posao. U Work-Out sistemima ova situacija je obrnuta, tj. menadžeri slušaju programere dok im objašnjavaju kakve bi korake trebalo preduzeti u datom trenutku da bi došli do željenih poboljšanja. Lean pristup favorizuje aforizam „Pronađi dobre ljude i pusti ih da rade svoj posao“.

Korisnik treba da ima opšti doživljaj sistema, tj. da osjeti integritet, kako se on reklamira, isporučuje, te kako je njegova primjena intuitivna i dr. Konceptualni integritet podrazumijeva da sistemski odvojene komponente rade zajedno kao cjelina, te da se uspostavi ravnoteža između fleksibilnosti, održavanja, efikasnosti i odziva. Ovo se može postići razumijevanjem domena problema i njegovim trenutnim rješavanjem, a ne sekvencijalnim. Protok informacija treba da bude dvosmijeran, od programera do kupca i obrnuto.

Softverski sistemi nisu samo skup dijelova softvera, već su oni i proizvod njihove interakcije. Nedostaci u softveru se akumuliraju tokom razvoja, od dijeljenja većih zadataka na manje, standardizacije u različitim fazama, te je potrebno otkriti uzrok i eliminisati ga. Lean mora biti shvaćen od strane svih članova tima prije korišćenja u nekom konkretnom slučaju.

3.Data-driven programiranje

Data-driven programiranje je programerska paradigma u kojoj se programski izvještaji opisuju podacima. Prilagođavanje apstraktnog tipa podataka metode objektno-orijentisanom tipu rezultuje dizajnu data-driven-a. Ovaj tip dizajna se ponekad koristi kod objektno-orijentisanog programiranja za definisanje klasa, tokom shvatanja dijela softvera. Osnovni principi data-driven programiranja su:

- Razdvajanje koda i podataka
- Podaci su vidljivi i mogu se mijenjati
- Promjena podataka može promijeniti logiku ili ponašanje sistema

Dok se prednosti i mane data-driven programiranja razlikuju u zavisnosti od primjene, postoji nekoliko ogromnih prednoti i mana ove paradigme. Funkcije i interfejsi mogu biti primijenjeni na sve objekte sa istim poljima podataka. Podaci se mogu grupisati u objekte ili entitete.

Iako data-driven programiranje ne sprečava spajanje podataka i funkcionalnosti, u nekim slučajevima data-driven može dovesti do lošeg objektno-orijentisanog dizajna, posebno ako se radi sa više apstraktnih podataka. Do ovog dolazi zato što su čisti data-driven objekti ili entiteti definisani na način na koji su prezentovani. Svaki pokušaj promjene strukture objekta trenutno bi izazvao prekid funkcija koje se oslanjaju na taj objekat. Kao trivijalan primjer može se predstaviti vožnja kroz niz od dvije raskrsnice (ulica sa raskrsnicom) gdje vozač mora skrenuti desno ili lijevo. Ako se raskrsnica predstavlja preko poštanskog koda (5-cifreni broj) i dva naziva ulica (tekstualni tip podataka) postoji mogućnost da se desi greška kada se dvije ulice sijeku više puta. Ovaj primjer može biti i pojednostavljen. Restruktuiranje podataka je prilično čest problem u softverskom inžinjerstvu, bilo da eliminiše greške, povećava efikasnost ili podržava nove funkcionalnosti. U ovim slučajevima dizajn zasnovan na odogovornosti može biti promovisan kao bolji pristup, gdje podaci i funkcionalnosti mogu biti zajedno u kombinaciji, tako da funkcije ne moraju da se suoče sa podacima koji ih reprezentuju.

Data-driven programiranje se ponekad miješa sa objektnom orijentacijom, kod koje bi organizacija podataka trebala biti centralizovana.

Postoje najmanje dvije razlike između objektne orijentacije i data-driven orijentacije:

- Kod data-driven programiranja podaci ne predstavljaju samo stanje nekog objekta, već zapravo definišu kontrolu toka programa.
- Primarni zadatak objektne orijentacije je enkapsulacija, dok je osnovni koncept kod data-drivena pisanje njmanjih mogućih fiksnih blokova koda.

4. ASP.NET Web forme

ASP.NET je tehnologija za razvoj Web aplikacija. Kombinuje najbolje servise koje omogućava ASP (Active Server Pages) i osobine koje obezbjeđuje CLR (Common Language Runtime), ali dodaje i nove funkcionalnosti. Rezultat je robustan, skalabilan i brz razvoj fleskibilnih aplikacija sa malo kodiranja.

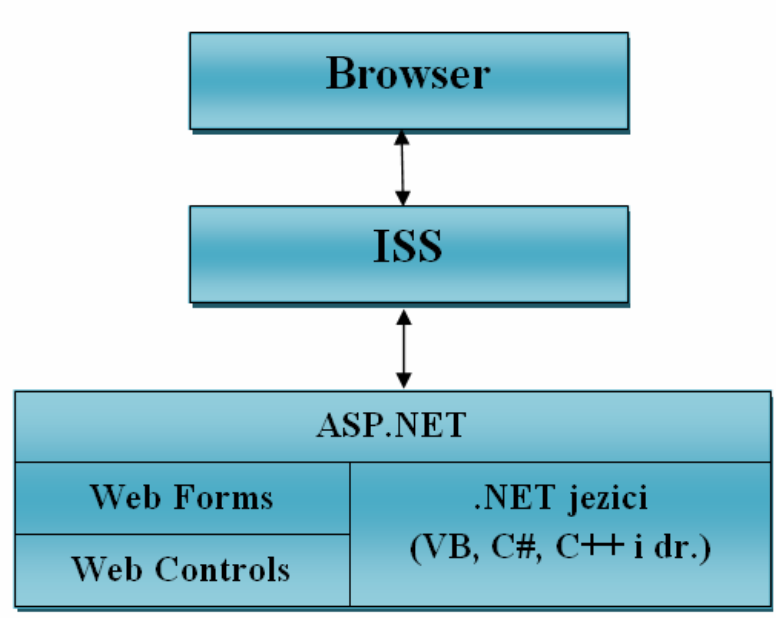
Web forme predstavljaju osnovni dio ASP.NET-a. One predstavljaju sastavni dio korisničkog interfejsa, te daju izgled i stil aplikaciji. Web forme su slične Windows formama, jer se kontrolišu preko opcija, metoda i događaja kao i Windows forme. Međutim, elementi korisničkog interfejsa se predstavljaju preko određenog jezika, na primje HTML-a. Microsoft Visual Studio .NET obezbjeđuje prevuci-i-pusti (drag-and-drop) interfejs za razvoj korisničkog interfejsa za Web aplikacije.

Web fome se sastoje od dvije komponente:

- Vizuelna komponenta (ASPX datoteka)
- Pozadinski kod (smješten u posebnoj datoteci)

Na slici 12 se vidi da su Web Forme sastavni dio ASP.NET aplikacije.

Slika 12 – Web forme kao dio ASP.NET-a



Web forme i ASP.NET su stvoreni da prevaziđu ograničenja ASP-a. Prednosti se odnose na:

- Razdvajanje HTML interfejsa od aplikacione logike
- Bogat skup serverskih kontrola koje browser može detektovati i proslijediti odgovarajućem jeziku, kao što je HTML
- Manje koda na serverskoj strani zbog data-binding kontrole
- Programerski model zasnovan na događajima
- Podrška kompajliranju sa različitih jezika
- Obezbjeđuje trećoj strani da kreira kontrolu koja stvara novu funkcionalnost

Vanjski izgled Web formi podsjeća na radnu površinu na koju se postavljaju različiti tipovi grafičkih komponenti. U stvarnosti one su mnogo više od toga. Svaka komponenta može sadržati svoje osobine, događaje i metode. Postoje dva tipa komponenti koje se mogu kreirati na korisničkom interfejsu, a to su HTML komponente i Web Form komponente.

U ovom radu akcenat je stavljen na komponente Web formi koje pristupaju bazi podataka, čitaju podatke iz baze, te ih prikazuju u adekvatnom obliku. Takođe, preko ovih komponenata se realizuje dodavanje, brisanje i izmjena podataka u bazi preko odogovarajućih komponenata Web forme.

4.1. Struktura ASPX stranice

ASPX stranica se sastoji od skupa komponenata i osobina tih komponenata koje definišu vizuelni izgled Web forme. Svaka ASPX stranica počinje sa linijom koja obavezno mora sadržati jezik u kome je napisan pozadinski kod, naziv fajla u kome se nalazi pozadinski kod i naziv aplikacije za koju je vezana stranica. Da bi kreirali serversku komponentu koja prikuplja podatke preko browsera neophodno je implementirati interfejs System.Web.UI.IPostBackEventHandler. U nastavku je prikazan primjer ove dvije linije koda.

```
<%@ Page Language="C#"  AutoEventWireup="true" CodeFile="Radnik.aspx.cs" Inherits="VebAplikacija"
Title="Radnik" %>
<%@ Implements Interface="System.Web.UI.IPostBackEventHandler"%>
```

Zaglavlje stranice pored ovih linija može sadržati i još neke dodatne. Nakon ovih linija slijedi deklarisanje SqlDataSource komponente ili komponenata. SqlDataSource je komponenta koja omogućava pristup bazi podataka. Sastoji se od identifikatora, connection stringa, komandi za selektovanje, dodavanje, brisanje i izmjenu, te parametara za ove komande.

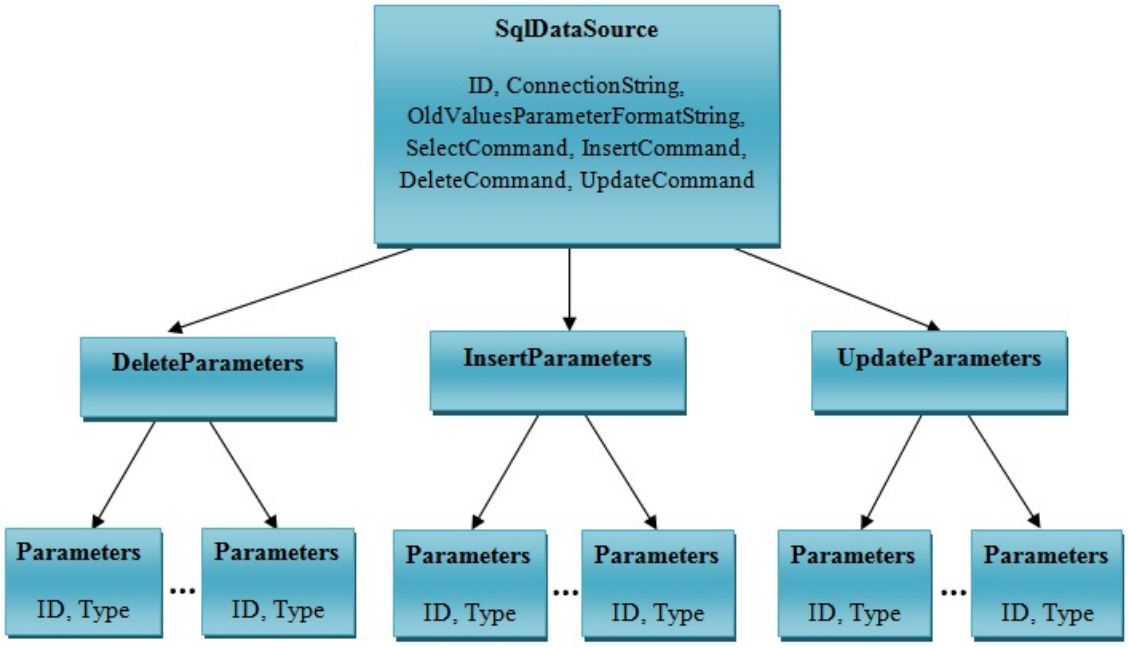
Identifikator predstavlja polje koje jednoznačno određuje dati SqlDataSource, a koristi se za pristup tom SqlDataSource-u.

ConnectionString je svojstvo koje obavezno mora biti unešeno u SqlDataSource, a glavna namjena mu je da prikaze lokaciju na kojoj se nalazi baza podataka, naziv baze podataka, parametre za pristup bazi, kao i druga opciona svojstva.

Komande koje se nalaze unutar deklaracije SqlDataSource komponente služe za pristup bazi podataka. One predstavljaju klasične SQL upite. SqlDataSource sadrži komande za selektovanje, dodavanje, brisanje i izmjenu, ali može sadržati i samo jednu od njih u zavisnosti od namjene konkretne komponente.

Parametri SQL komandi se moraju poklapati sa SQL komandama. Ukoliko postoji konkretna komanda moraju postojati parametri za tu komandu. U skladu sa SQL komandama mogu se identifikovati različite vrste parametara, a to su SelectParameters, DeleteParameters, InsertParameters i UpdateParameters. Izgled SqlDataSource komponente dat je na slici 13.

Slika 13 – Prikaz SqlDataSource komponente sa atributima i podkomponentama



U nastavku je prikazan konkretan primjer SqlDataSource komponente.

```
<asp:SqlDataSource ID="RadnikSRC" runat="server"
ConnectionString="Server=.\SQLExpress;AttachDbFilename=C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\Firma.mdf;Database=Firma; Trusted_Connection=Yes; Integrated
Security=True;Connect Timeout=30"
OldValuesParameterFormatString="original_{0}" OnInserting="RadnikSRC_Inserting"
DeleteCommand="DELETE FROM [Radnik] WHERE [ID] = @ID"
InsertCommand="INSERT INTO[Radnik]([Ime], [Prezime], [Telefon], [Staz]) VALUES (@Ime, @Prezime,
@Telefon, @Staz)"
SelectCommand="SELECT [Ime], [Prezime], [Telefon], [Staz] FROM [Radnik]"
UpdateCommand="UPDATE [Radnik] SET  [ID] = @ID, [Ime] = @Ime, [Prezime] = @Prezime, [Telefon] =
@Telefon, [Staz] = @Staz">
        <DeleteParameters>
                <asp:Parameter Name="ID" Type="Int32" />
        </DeleteParameters>
        <InsertParameters>
                <asp:Parameter Name="Ime" Type="String" />
                <asp:Parameter Name="Prezime" Type="String" />
                <asp:Parameter Name="Telefon" Type="String" />
                <asp:Parameter Name="Staz" Type="Int32" />
        </InsertParameters>
        <UpdateParameters>
                <asp:Parameter Name="@ID" Type="Int32" />
                <asp:Parameter Name="Ime" Type="String" />
                <asp:Parameter Name="Prezime" Type="String" />
                <asp:Parameter Name="Telefon" Type="String" />
                <asp:Parameter Name="Staz" Type="Int32" />
        </UpdateParameters>
</asp:SqlDataSource>
```

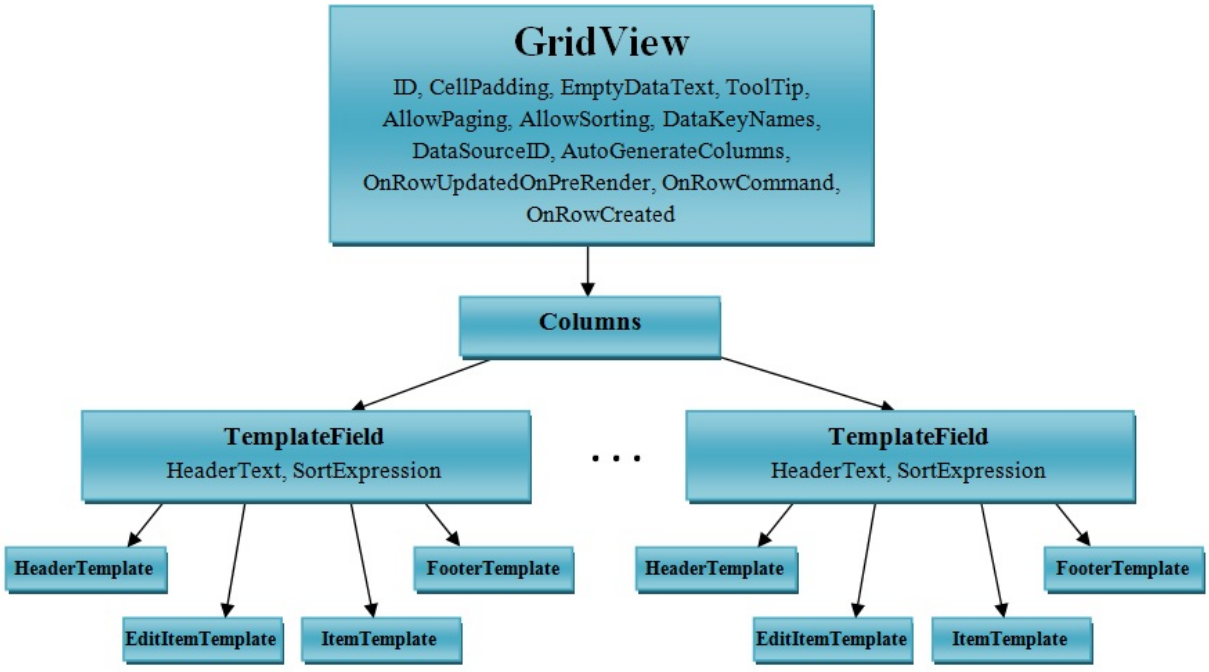
Nakon deklarisanja jedne ili više SqlDataSource komponente, mogu se deklarisati Validacione grupe, koje treba da provjere podatke za unos i prepravku, a koje se kasnije dodaju određenim komponentama. Takođe, opciona varijanta je dodavanje komponente LinkedButton DodajPrazno koja treba da ubaci u bazu jedan prazan zapis. Moguće je definisati i labelu koja će služiti za prikaz grešaka. U nastavku je dat primjer ovih komponenata.

```
<asp:Label ID="ExceptionDetails" runat="server" Text="" Font-Size="X-Large"/>
<asp:LinkButton ID="DodajPrazno" runat="server" style="margin-left:10px" OnClick="DodajPrazno_Click"
Text="Dodaj prazno (bez podataka)"></asp:LinkButton>
<asp:ValidationSummary ID="VS1" runat="server" ShowMessageBox="True" ShowSummary="False"
ValidationGroup="prepravka"/>
<asp:ValidationSummary ID="VS2" runat="server" ShowMessageBox="True" ShowSummary="False"
ValidationGroup="dodavanje"/>
```

Naredni korak je deklarisanje komponente za prikaz podataka. Najčešće se koristi GridVew komponenta sama ili u kombinaciji sa DetailView komponentom ukoliko tabela ima velik broj atributa. Osobine koje se navode pri deklaraciji GridView komponente su identifikator, naziv SqlDataSource komande sa kojom je povezan GridView, opcije koje omogućavaju sortiranje i straničenje, kao i različite vrste događaja i nazivi metoda koje se nalaze u pozadinskom kodu, a treba da odgovore na te događaje. Moguće je definisiati neki od stilova za prikaz GridView komponente, koji će definisati koje je boje određeni dio GridView komponente, kakvi fontovi se koriste, ali i druge opcione osobine. Unutar deklaracije GridView komponente potrebno je deklarisati kolone, obično preko TemplateField komponenti. Jedna TemplateField komponenta se sastoji od sljedećih podkomponenti:

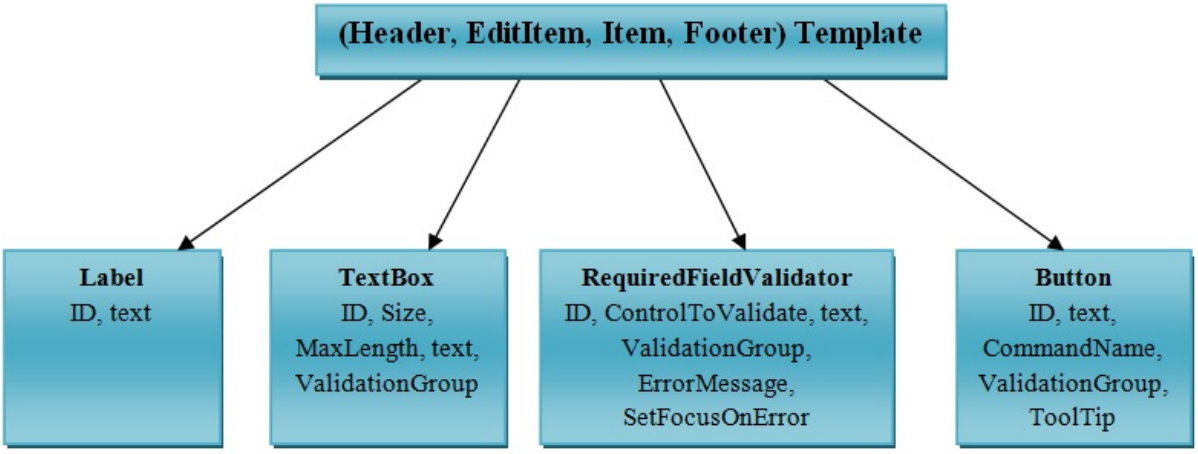
- EditItemTemplate
- ItemTemplate
- FooterTemplate

Slika 14 – Prikaz GridView komponente sa atributima i podkomponentama



Svaka od template komponenti (HeaderTemplate, EditItemTemplate, ItemTemplate, FooterTemplate) može imati podkomponente: Label, TextBox, RequiredFieldValidator, Button i dr.. Na slici 15 prikazana je struktura Template komponente.

Slika 15 – Prikaz Template komponente podkomponentama i njihovim atributima



U nastavku je prikazan konkretan primjer GridView komponente koja sadrži opcije za dodavanje, brisanje i izmjenu zapisa. Sadrži također i polje za prikaz ukupnog broja zapisa.

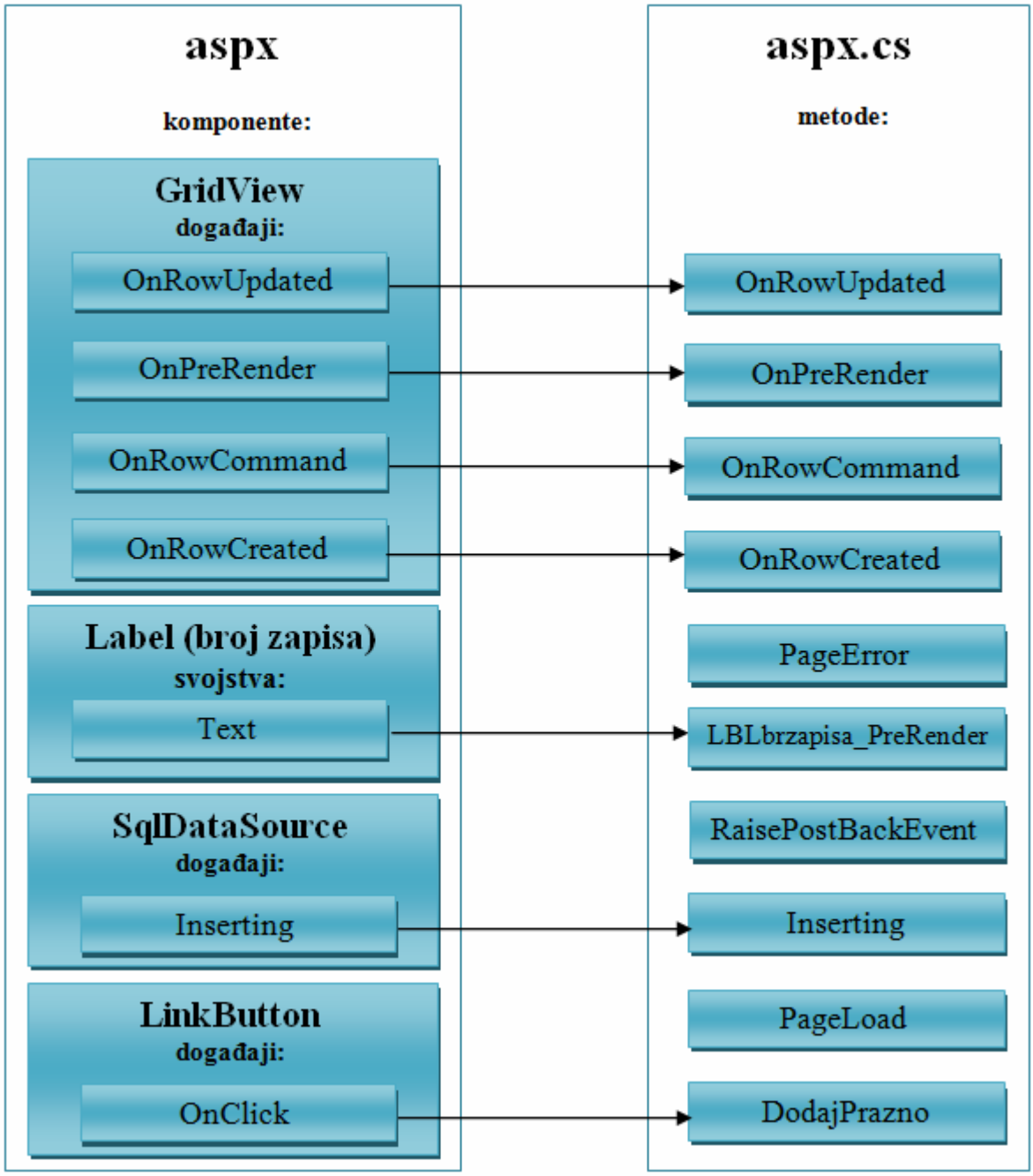
```
<asp:GridView ID="gvRadnik" runat="server"  EmptyDataText="Ovdje nema podataka" ShowFooter="True"
ToolTip="Unos, prepravka i brisanje." AllowPaging="True" AllowSorting="True" DataKeyNames="ID"
DataSourceID="RadnikSRC" AutoGenerateColumns="False" OnRowUpdated="gvRadnik_RowUpdated"
OnPreRender="gvRadnik_PreRender" OnRowCommand="gvRadnik_RowCommand"
OnRowCreated="gvRadnik_RowCreated" CellPadding="4" ForeColor="#333333" GridLines="None">
  <RowStyle BackColor="#FFFBD6" ForeColor="#333333" />
  <Columns>

   <asp:TemplateField>
        <EditItemTemplate>
          <asp:Button ID="BTN1" runat="server" CausesValidation="True" ToolTip="[Snimi prepravke.]"
           CommandName="Update" Text="Snimi" ValidationGroup="prepravka"/>
           <asp:Button ID="BTN2" runat="server" CausesValidation="False" CommandName="Cancel"
           Text="Odustani" ToolTip="[Odustani od prepravki.]" />
        </EditItemTemplate>
        <ItemTemplate>
          <asp:Button ID="BTN3" runat="server" CausesValidation="False" CommandName="Edit"
           Text="Prepravi"  ToolTip="[Prepravi podatke o zvanju.]" />
        </ItemTemplate>
        <FooterTemplate>
          <asp:Button ID="BTN4" runat="server" Text="Додај" CommandName="Insert"
           ToolTip="[Dodaj unijete podatke u bazu podataka.]" ValidationGroup="dodavanje"/>
        </FooterTemplate>
  </asp:TemplateField>

  <asp:TemplateField HeaderText="ID" InsertVisible="False" SortExpression="ID">
       <ItemTemplate><asp:Label ID="LBLID" runat="server" Text='<%# Bind("ID") %>'/></ItemTemplate>
  </asp:TemplateField>
```

```
    <asp:TemplateField HeaderText="Ime" InsertVisible="True" SortExpression="Ime">
            <EditItemTemplate>
                <asp:TextBox ID="TB1" Size="10" MaxLength="10"  runat="server" Text='<%# Bind("Ime") %>'
                ValidationGroup="prepravka"  />
                 <asp:RequiredFieldValidator ID="RFV1" runat="server" ValidationGroup="prepravka"
                ControlToValidate="TB1" ErrorMessage="Morate unijeti vrijednost u polje." Text="*"
                SetFocusOnError="true"  />
            </EditItemTemplate>
            <ItemTemplate>
                 <asp:Label ID="LBL1" runat="server" Text='<%# Bind("Ime")%>'/>
            </ItemTemplate>
            <FooterTemplate>
                    <asp:TextBox ID="TBIme" runat="server" Size="10" MaxLength="10"
                      ValidationGroup="dodavanje" />
                     <asp:RequiredFieldValidator ID="RFV1" runat="server" ValidationGroup="dodavanje"
                    ControlToValidate="TBIme" Text="*" ErrorMessage="Morate unijeti vrijednost u polje."
                    SetFocusOnError="true" />
              </FooterTemplate>
</asp:TemplateField>

 <asp:TemplateField ShowHeader="False">
    <ItemTemplate>
            <asp:Button ID="BTN7" runat="server" CausesValidation="False" CommandName="Delete"
            Text="Obriši"  OnClientClick="return confirm('Jeste li sigurni da želite obrisati?');"
            ToolTip="[Obriši]"/>
    </ItemTemplate>
    <FooterTemplate>
            <asp:Label ID="LBLbrzapisa" BorderStyle="Double" Font-Bold="True" ToolTip="Ukupan broj zapisa"
            runat="server" OnPreRender="LBLbrzapisa_PreRender" />
    </FooterTemplate>
 </asp:TemplateField>

 </Columns>
</asp:GridView>
```

4.2. Struktura pozadinskog koda

Kao što je ranije spomenuto pozadinski kod sadrži metode koje treba da daju funkcionalnost web formi. Postoje dvije varijacije metoda u pozadinskom kodu, jedna je da događaj poziva odgovarajuću metodu, a druga je da metoda poziva (koristi odgovarajuće svojstvo) Na slici 16 prikazana je struktura ASPX komponenti i njima pripadajućih događaja, te je prikazana struktura metoda pozadinskog koda i napravljena veza između događaja i njima ekvivalentinih metoda.

Slika 16 – Struktura ASPX stranice i pozadinskog koda



Prva metoda u pozadinskom kodu je Page_Load. Ova metoda se izvršava pri pokretanju stranice. Obično sadrži postavljanje inicijalnih vrijednosti promjenljivih, kao i prikupaljanje informacija o tome koji je korisnik prijavljen na sistem.

Sljedeća implementirana metoda je Page_Error koja izvršava naredbu Server.GetLastError().

Metoda PreRender se pokreće nakon što su na stranici kreirane sve komponente koje su zahtijevane da bi se stranica rendovala, uključujući i podkomponente složenih komponenata. Stranica poziva EnsureChildControls za svaku komponentu na stranici. Stranica poziva događaj PreRender za komponentu, a zatim rekurzivno radi isto za svaku podkomponentu. PreRender pojedine komponente se javlja nakon PreRender događaja stranice. Kada je omogućeno straničenje dodatni red koji se zove red straničenja je omogućen i automatski prikazan u GridView kontroli. Realizacija metode PreRender omogućava red straničenja. U nastavku su prikazane konkretne metode PageLoad i PreRender.

```
protected void Page_Error(Object sender, System.EventArgs e)
 {
   throw Server.GetLastError();
 }
 protected void gvRadnik_PreRender(object sender, EventArgs e)
 {
    if (this.gvRadnik.BottomPagerRow != null)
          this.gvRadnik.BottomPagerRow.Visible = true;
 }
```

Sljedeća PreRender metoda je LBLbrzapisa_PreRender. Ova metoda upisuje broj zapisa u labelu LBLbrzapisa. Metoda LBLbrzapisa_PreRender prikazana je u nastavku.

```
protected void LBLbrzapisa_PreRender(object sender, EventArgs e)
{
    SqlConnection con = new SqlConnection ("Server=.\SQLExpress;AttachDbFilename=C:\Program
    Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\Firma.mdf;Database=Firma; Trusted_Connection=Yes;
    Integrated Security=True;Connect Timeout=30");
    SqlCommand cmd = new SqlCommand("SELECT COUNT(*) FROM Radnik", con);
    con.Open();
    int brojredova = (int)cmd.ExecuteScalar();
    con.Close();
    Label l = (Label)gvRadnik.FooterRow.FindControl("LBLbrzapisa");
    l.Text = "--" + brojredova.ToString() + "--";
 }
```

Nakon PreRender metoda realizovana je metoda RaisePostBackEvent. Stranica poziva ovu metodu kada se desi postback. Ovaj poziv se dešava u životnom ciklusu stranice, nakon učitavanja stranice, ali prije PreRnder metoda. Realizacija ove metode prikazana je u nastavku.

```
public void RaisePostBackEvent(string eventArgument)
 {
    GridViewSelectEventArgs e = null;
    int selectedRowIndex = -1;
    if (!string.IsNullOrEmpty(eventArgument))
    {
        string[] args = eventArgument.Split('$');
        Int32.TryParse(args[0], out selectedRowIndex);
        e = new GridViewSelectEventArgs(selectedRowIndex % 10);
        gvRadnik.EditIndex = ((GridViewSelectEventArgs)e).NewSelectedIndex;
    }
 }
```

Prije nego što se GridView komponenta renduje, GridViewRow objekat mora biti kreiran za svaki red u komponenti. RowCreated metoda se pokreće nakon kreiranja svakog reda u GridView komponenti. Ova metoda obezbjeđuje izvršavanje prilagođene rutine, kada se desi RowCreated događaj. U ovoj metodi se svakom redu može dodati sadržaj. Moguće je dodjeljivanje određenih opisnih slika, kao što je na primjer dodavanje slike koja definiše smijer sortiranja.

RowUpdated metoda se pokreće kada se klikne na dugme Izmijeni, tj. kada se pozove komanda Update. Ukoliko se desi greška pri izmjeni nekog zapisa. RowUpdated metoda treba da prikaže grešku koja se desila. RowUpdated metoda je prikazana u nastavku.

```
protected void gvRadnik_RowUpdated(object sender, GridViewUpdatedEventArgs e)
{
    if (e.Exception != null)
    {
        ExceptionDetails.Text = "Došlo je do greške pri snimanju pregledajte ukucane podatke.";
        e.ExceptionHandled = true;
        e.KeepInEditMode = true;
    }
```

RowCommand je metoda koju pokreće istoimeni događaj kada se klikne na dugme u komponenti GridView. Komandna dugmad unutar GridView kontrole mogu pozvati neku od ugrađenih funkcija te kontrole. U nastavku je prikazana realizacija ove metode, gdje se klikom na dugme Insert poziva komanda Insert imenovane SqlDataSource komponente.

```
protected void gvRadnik_RowCommand(object sender, GridViewCommandEventArgs e)
 {
     if (e.CommandName == "Insert" && Page.IsValid)
         RadnikSRC.Insert();
 }
```

Naredna metoda je Inserting. Ovo je metoda koja se izvršava prije komande ubacivanja, a obično je zadužena za obavljanje operacija inicijalizacije, provjeru vrijednosti parametara, ili promjenu vrijednost parametara prije nego što SqlDataSource izvrši komandu Insert. Veza sa skladištem podataka nije otvorena u trenutku kada se ova metoda izvršava. U nastavku je prikazan konkretan primjer ove metode.

```
protected void RadnikSRC_Inserting(object sender, SqlDataSourceCommandEventArgs e)
 {
      TextBox dod1 = (TextBox)gvRadnik.FooterRow.FindControl("TBIme")
      e.Command.Parameters["@Ime"].Value = dod1.Text;
      TextBox dod2 = (TextBox)gvRadnik.FooterRow.FindControl("TBPrezime")
      e.Command.Parameters["@Prezime"].Value = dod2.Text;
      TextBox dod3 = (TextBox)gvRadnik.FooterRow.FindControl("TBTelefon")
      e.Command.Parameters["@Telefon"].Value = dod3.Text;
      TextBox dod4 = (TextBox)gvRadnik.FooterRow.FindControl("TBStaz")
      e.Command.Parameters["@Staz"].Value = dod4.Text;
 }
```

Metoda DodajPrazno ima za cilj da doda prazan zapis u tabelu, tj. da izvrši insert komandu nad bazom podatka. Da bi se upit izvršio neophodno je napraviti objekat SqlConnection koji kao parametar ima ConnectionString objekat, a zatim napraviti objekat SqlCommand koji kao parametar sadrži konkretnu Insert komandu i objekat SqlConnection. Nakon toga potrebno je uspostaviti vezu sa bazom podataka, izvršiti upit, te prekinuti vezu. Da bi promjene bile vidljive u objektu GridView neophodno je izvšiti metodu DataBind() nad datom GridView komponentom. Metoda DodajPrazno je prikazana u nastavku.

```
protected void DodajPrazno_Click(object sender, EventArgs e)
 {
    SqlConnection con = new SqlConnection("Server=.\SQLExpress;AttachDbFilename=C:\Program
     Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\Firma.mdf;Database=Firma;Trusted_Connection=Yes;
     Integrated Security=True;Connect Timeout=30");
    SqlCommand cmd = new SqlCommand("INSERT INTO Radnik Default Values", con);
    try
    {
       con.Open();
       cmd.ExecuteNonQuery();
    }
    finally
    {
       con.Close();
    }
 gvRadnik.DataBind();
 }
```

5. Opis AutoGen aplikacije

AutoGen aplikacija je aplikacija koja služi za automatsko generisanje ASP.NET web forimi. Generator aplikacije predstavlja softversku komponenu koja služi za geneisanje druge softverske komponente.

Svaki generator na osnovu dobijenih ulaznih podataka i određenih pravila i procedura generiše izlaz. Ova operacija prikazana je na slici 17.

Slika 17 – Opšti model generatora aplikacija

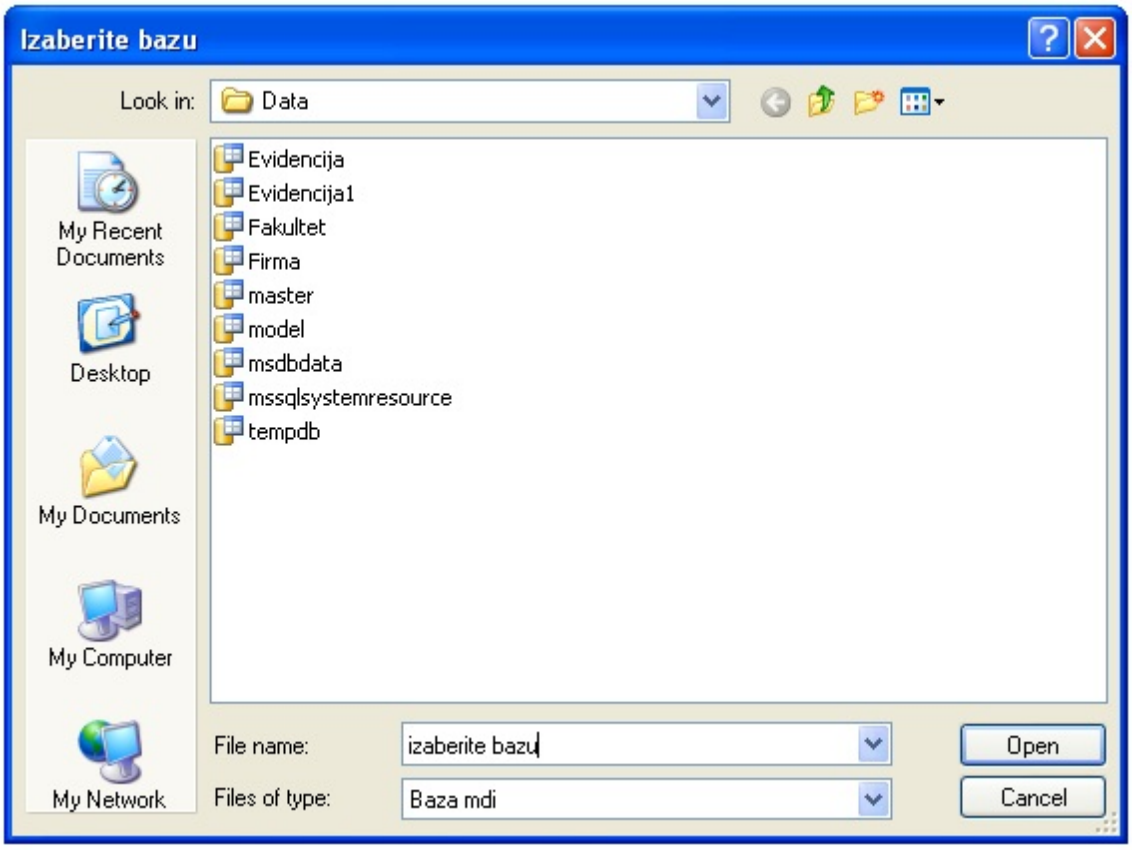
Kao ulaz za AutoGen aplikaciju koriste se podaci iz baze podataka, ali i podaci i svojstva koja se unose u toku izvršavanja same aplikacije. Kod ove aplikacije kao baza podataka se koristi SQL Server baza podataka, a izlazni rezultat je ASP.NET web forma koja se sastoji od ASPX stranice i pozadinskog koda. Aplikacija je razvijena na .NET platformi u jeziku C#.

Tok generisanja jedne ASP.NET web forme AutoGen aplikacijom teče kroz pet faza. U svakoj fazi se od korisnika prikupljaju određeni korisnički zahtjevi, kao i informacije o ulaznim i izlaznim putanjama. Korisnik ima mogućnost manipulacije izlaznim datotekama kroz praćenje koraka, ali i nakon što je kôd generisan.

AutoGen aplikacija je projektovana tako da ima mogućnost generisanja tri različita slučaja. Prvi slučaj predstavlja generisanje jedne GridView komponente na web formi, koja dobija podatke iz jedne tabele, tj. iz jednog SqlDataSource objekta. Drugi slučaj je kada se generiše jedna GridView komponenta, ali tako da ona koristi podatke iz više SqlDataSource objekata. Moguće je korišćenje maksimalno 5 eksternih izvora, tj. SqlDataSource objekata. Treći slučaj podrazumijeva korišćenje kombinacije GridView komponente sa komponentom DetailView, a da se pri tom koristi više SqlDataSource objekata. U nastavku će biti prikazan tok generisanja trećeg slučaja.

Pri pokretanju AutoGen aplikacije pojavljuje se početna forma u koju je potrebno unijeti putanju do baze podataka. Putanja se unosi klikom na dugme „Pronađi bazu“. Nakon klika na dugme prikazuje se dijalog za otvaranje baze kao na slici 18.

Slika 18 – Dijalog za otvaranje baze



Nakon izbora baze neophodno je popuniti polja „Naziv baze“, u koje se upisuje logički naziv baze podataka, te „Naziv tabele“ u koje se unosi naziv osnovne tabele iz izabrane baze. U naredno polje potrebno je unijeti naziv web aplikacije za koju se kreira web forma. Klikom na dugme „Otvori direktorijum“ otvara se dijalog za izbor izlazne putanje kao na slici 19.

Slika 19 – Dijalog za izbor izlazne putanje

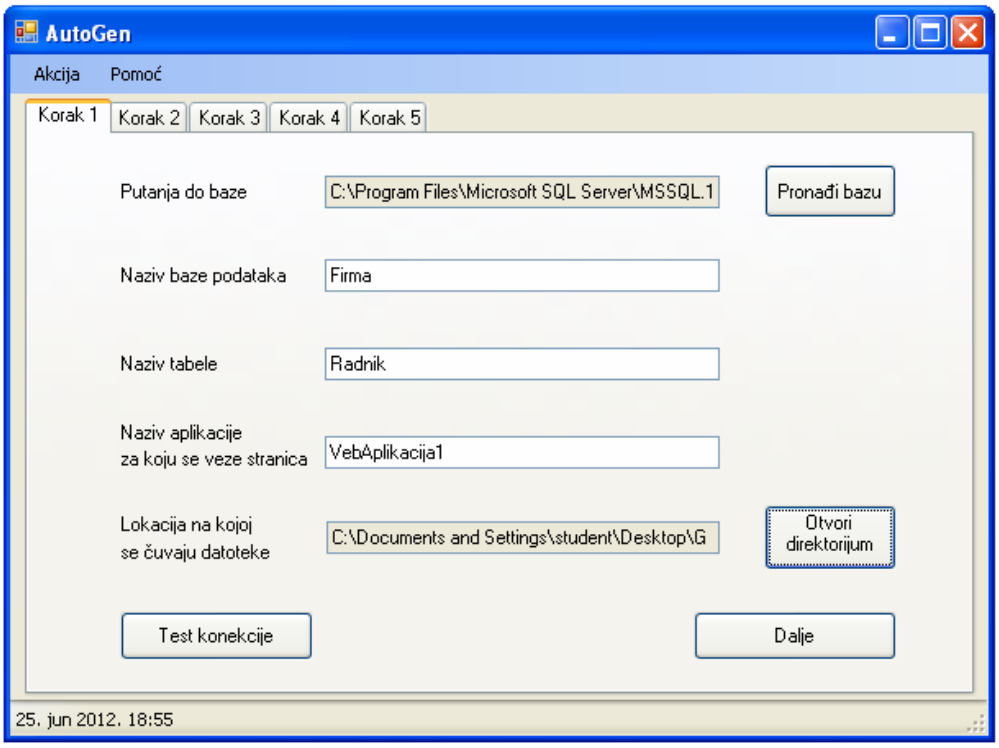
Nakon unosa ovih parametara moguće je provjeriti vezu sa bazom. Ukoliko kliknete na dugme „Test konekcije“, i ukoliko su svi parametri ispravno unešeni prikazaće se forma u kojo su ispisani zapisi iz tabele sa kojom je veza uspostavljena. Ukoliko dođe do greške pri povezivanju korisnik će biti prikladno obaviješten o prirodi greške. Na slici 20 prikazan je prozor „Test konekcije“.

Slika 20 – Test veze sa tabelom u bazi

	ID	Ime	Prezime	Telefon	Staz
▶	1	Marko	Markovic	065/258-332	2
	2	Simo	Simic	065/256-652	3
	3	Petar	Petrovic	066/856-658	5
	4	Lazo	Lazic	062/565-958	3
	5	Aco	Acic	061/896-652	5
	6	Petko	Petkovic	065/859-985	5
*					

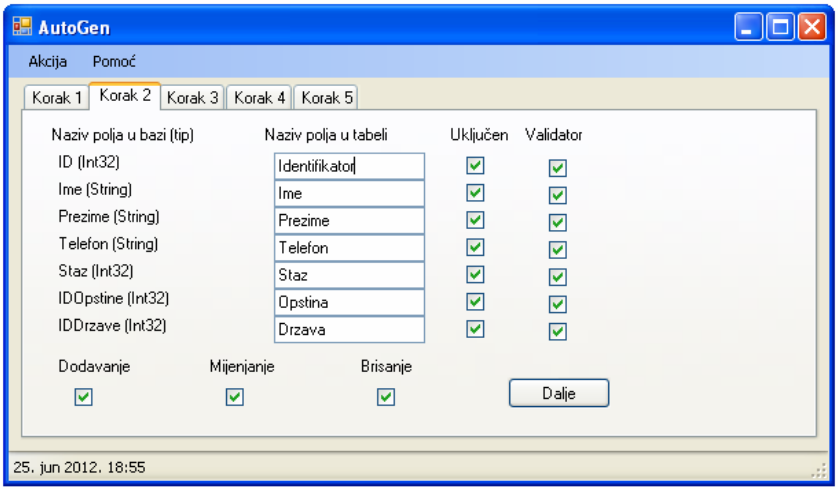
Nakon zatvaranja prozora „Test konekcije“, aplikacija izgleda kao na slici 21.

Slika 21 – Prikaz koraka 1



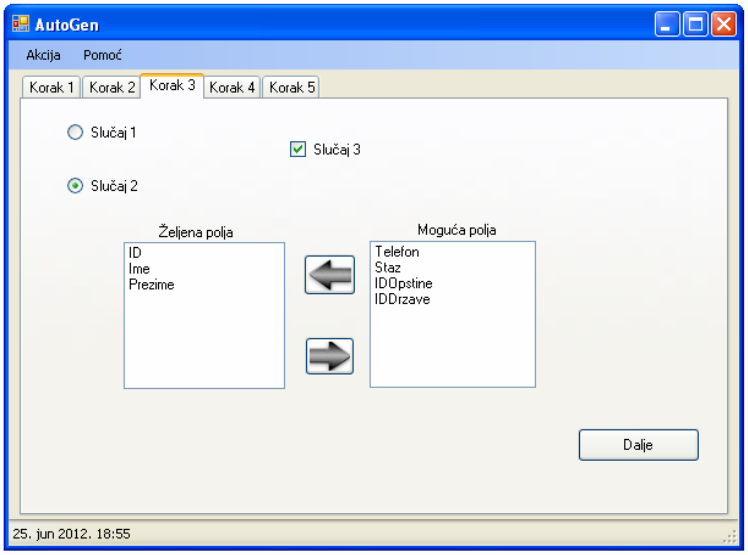
Ukoliko je veza sa bazom uspješna može se preći na sljedeći korak, klikom na dugme „Dalje“. Ovim se prelazi na Korak 2. Korak 2 dinamički kreira polja u zavisnosti od podataka iz tabele. U ovom koraku se ispisuju svi nazivi atributa iz izabrane tabele, tipovi tih atributa, kao i polje naziv u tabeli koje definiše kakav će naziv to polje imati na web formi. Ukoliko se naziv ne promijeni, ostaće isti kao što je u bazi. Opcija „Uključeno“ označava da li će polje biti prikazano na web formi ili ne. Ukoliko je ova opcija štiklirana polje će biti prikazano, a ako nije, polje neće biti prikazano. Podrazumijevano su sva polja uključena. Sljedeća opcija „Validator“ definiše da li će se nad tim poljem raditi validacija. Ukoliko je polje štiklirano validacija će biti uključena za to polje. Kao i kod prethodnog polja, validacija je podrazumijevano uključena za sva polja. U donjem dijelu se nalaze opcije „Dodavanje“, „Mijenjanje“ i „Brisanje“. Ukoliko je neka od ovih opcija isključena odgovarajuće dugme za tu opciju na web formi neće biti prikazano. Ukoliko je na primjer ugašena opcija „Brisanje“, na web formi neće postojati dugme za brisanje. Izgled koraka 2 je prikazan na slici 22.

Slika 22 – Prikaz koraka 2



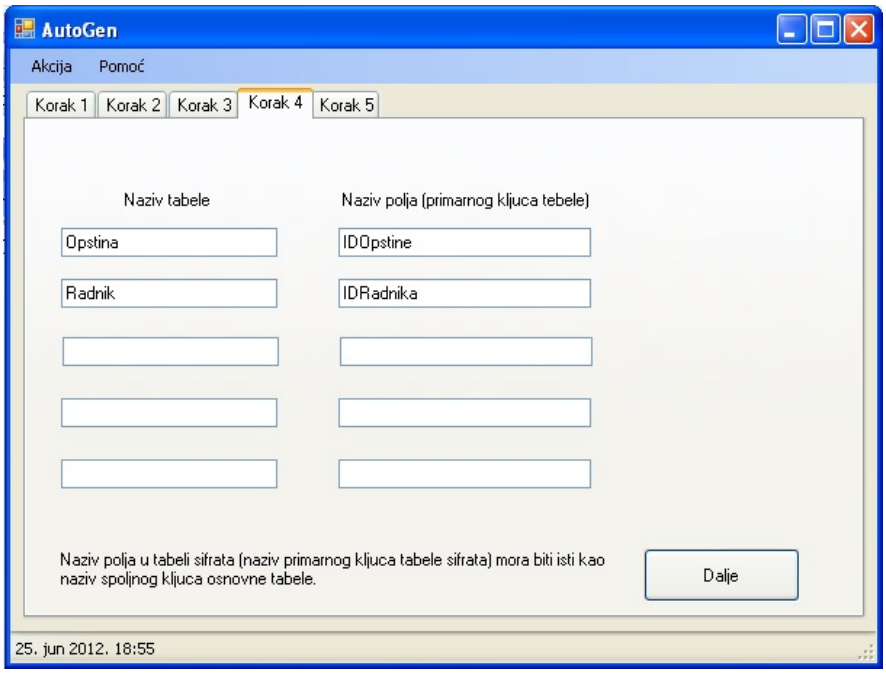
Klikom na dugme dalje prelazi se na korak 3. Korak 3 nudi izbor između tri različita slučaja koja su ranije objašnjena. Ukoliko se izabere slučaj 1 ili slučaj 1 u kombinaciji sa slučajom 3, u ovom koraku nema više podešavanja i može se preći na sljedeći. Ukoliko se izabere slučaj 2 u kombinaciji sa slučajom 3 onda je neophodno iz liste mogućih polja izabrati željena polja koja će biti prikazana na komponenti GridView. Ova opcija se preporučuje ukoliko tabela ima previše atributa tako da rezolucija ekrana ne podržava prikaz svih atributa bez potrebe za skrolovanjem. Korak 3 prikazan je na slici 23.

Slika 23 – Izgled koraka 3



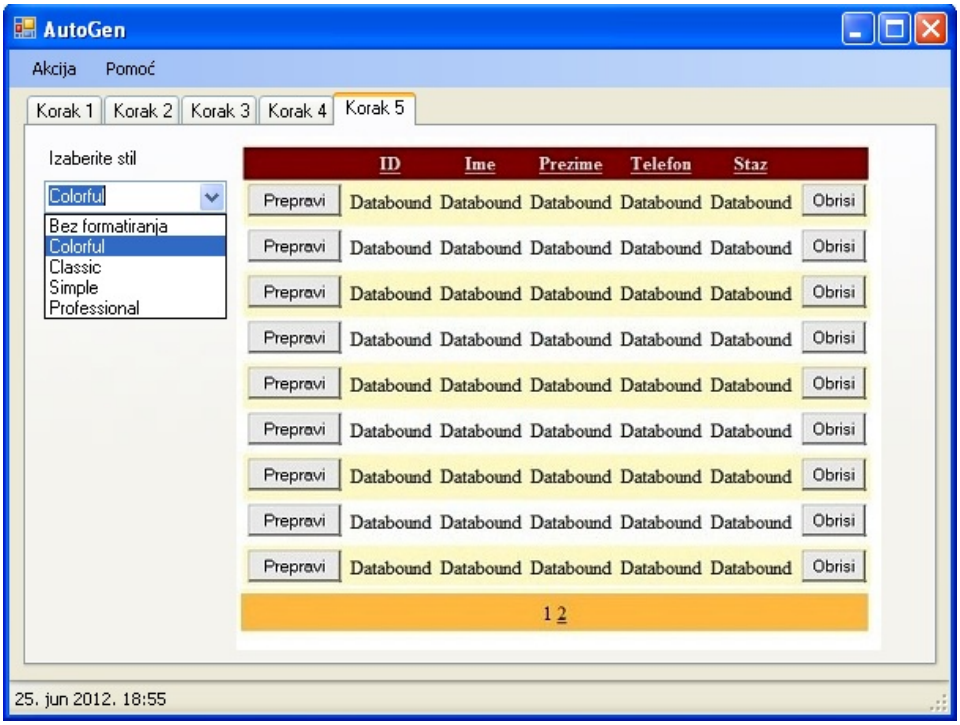
Klikom na dugme „Dalje“, prikazuje se korak 4. U ovom koraku potrebno je unijeti nazive eksternih tabela, tj. tabela sa kojima se iz osnovne tabele uspostavlja veza i naziv primarnog ključa pojedine tabele. Neophodno je da tabele budu u obliku šifrarnika, tj. da sadrže atribut primarni ključ i jedan atribut koji opisuje dati slog. Takođe je neophodno da naziv primarnog ključa eksterne tabele bude isti kao i naziv spoljnog ključa u osnovnoj tabeli. Izgled koraka 4 dat je na slici 24.

Slika 24 – Izgled koraka 4



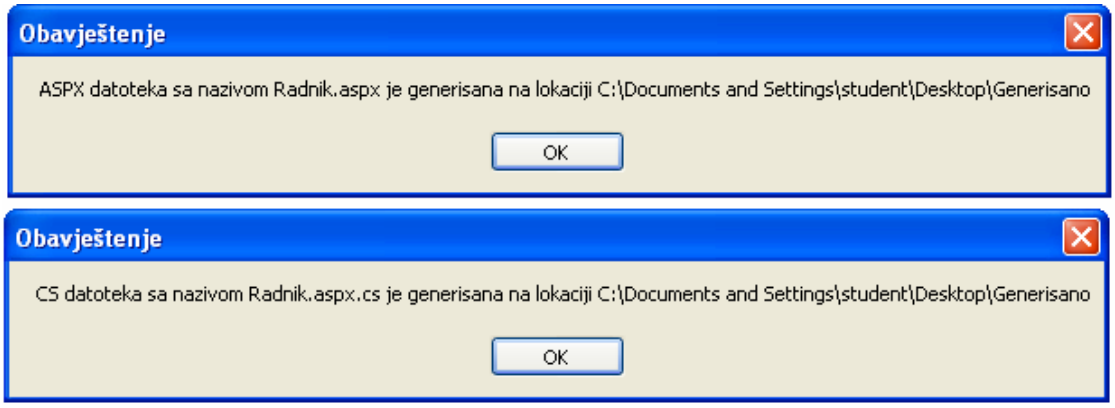
Nakon podešavanja parametara eksterne tabele može se preći na sljedeći korak. Sljedeći korak, koji je ujedno i posljednji korak je određivanje izgleda web forme. U AutoGen aplikaciju je ugrađeno 5 stilova. Svaki od stilova definiše određene osobine web forme. Definisane su boje pozadine, boje zaglavlja, podnožja, različite boje parnih i neparnih slogova, kao i položaj i boja reda za straničenje. Jedan od ponuđenih stilova je „Bez formatiranja“, koji ne dodaje nikakve efekte izgledu forme. Ovi stilovi su predefinisani stilovi za izgled web forme. Na slici 25 dat je izgled koraka 5 sa izabranom „ColorFul“ stilom iz padajuće liste stilova.

Slika 25 – Izgled koraka 5



Kada se klikne dugme „Generiši“ AutoGen aplikacija počinje sa generisanjem koda. Ukoliko je sav proces generisanja protekao uspješno na ekranu će se prikazati poruke o uspješnosti kao na Slici 26.

Slika 26 – Prikaz poruka uspješnosti

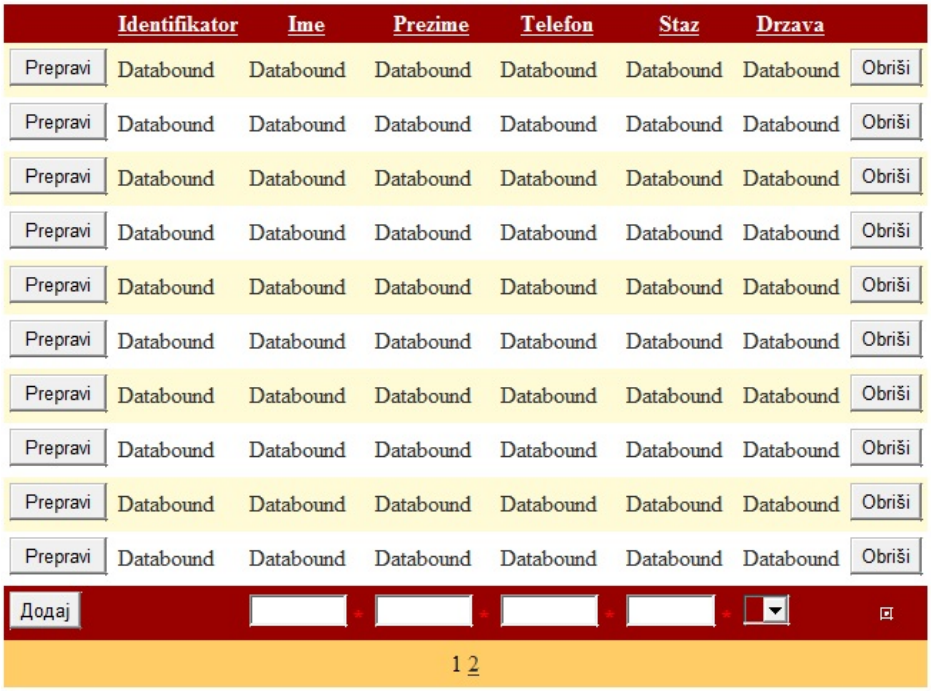


Aplikacija AutoGen generiše dvije datoteke na zadatoj lokaciji, datoteku sa ekstenzijom aspx i datoteku sa ekstenzijom aspx.cs.

Izgled web forme koja je generisana, prikazan je na slici 27, a na slici 28 je prikazan izgled web forme ukoliko se koristi slučaj 1 u kombinaciji sa slučajem 3.

Slika 27 – Izgled generisane web forme u Visual Studiu

Slika 28 – Izgled generisane web forme u Visual Studiu ako se koristi GridView prikaz iz više tabela



6. Zaključak

U prvom dijelu ovog rada opisane su metodologije razvoja sofvera. Napravljen je pregled najpoznatijih metoda i modela razvoja softvera. Metodologije koje su opisane u radu su:

- Rappid Application Development
- Model vodopada
- Spiralni model
- Rational jedinstveni proces razvoja
- Larmanova metoda
- Razvoj vođen testovima
- Agilna metoda razvoja softvera
- Scrum
- Ekstremno programiranje

U nastavku rada opisana je Data-driven programerska paradigma, koja predstavstavlja razvoj izvještaja korišćenjem podataka.

U trećem dijelu rada opisana je struktura ASP.NET web forme, koja se sastoji od ASPX stranice i pozadinskog koda. Kod ASPX stranice akcenat je stavljen na prikaz GridView i SqlDataSource komponente, dok je u opisu pozadinskog koda akcenat stavljen na opis metoda koje daju funkcionalnost GridView komponenti.

Sav ovaj opis je napravljen kao uvod u aplikaciju AutoGen, koja je rađena metodološki, podržava Data-driven programersku paradigmu, te služi za generisanje ASP.NET web formi. AutoGen aplikacija se koristi za automatizovano generisanje ASPX stranice i pozadinskog koda. Kao metodologija pri izradi ovog rada izabrana je metodologija Rappid Application Development. Aplikacija se mogla razviti i sa nekom drugom metodologijom, ali je ova odabrana kao najpogodnija.

Neke od prednosti korišćenja AutoGen aplikacije su:

- skraćivanje vremena razvoja aplikacije
- povećanje kvaliteta programskog koda

- eliminacija monotonije u radu
- mogućnost ponovne upotrebe
- mogućnost višestruke upotrebe
- smanjenje broja grešaka

AutoGen aplikacija predstavlja konceptualnu aplikaciju, a ne komercijalni proizvod. Ona bi mogla postati komercijalna uz određena proširenja i poboljšanja.

U narednim verzijama AutoGen aplikacije sigurno će biti riječi o proširenju funkcionalnosti, tj. proširenju AutoGen-a tako da on generiše čitavu ASP.NET aplikaciju. To bi onda bio jedan jako atraktivan softverski alat koji bi sigurno našao svoj komercijalni put.

7. Zahvalnica

Ovom prilikom se zahvaljujem svom mentoru doc. dr Đorđu Babiću, na velikoj podršci. Posebno se zahvaljujem asistentu Dragoljubu Pilipoviću za pomoć i stručne savjete koje mi je pružio pri izradi ovog rada.

8. Literatura

[1] Vlajic, S. (2011), Projektovanje softvera (Skripta), FON, Beograd.

[2] Stanilović, M. (2009), Razvoj JEE aplikacije primenom ekstremnog programiranja (Master rad), FON, Beograd.

[3] Lazetić, S. (2010), Razvoj generatora Spring aplikacija primenom Freemaker šablona i Hibernate okvira (Master rad), FON, Beograd.

[4] Wikipedia, "Software development methodology", http://en.wikipedia.org/wiki/Software_development_methodology, pristupljeno 21.06.2012.

[5] Wikipedia, "Rappid Application Development", http://en.wikipedia.org/wiki/Rapid_application_development, pristupljeno 21.06.2012.

[6] Wikipedia, "Waterfall model", http://en.wikipedia.org/wiki/Waterfall_model, pristupljeno 21.06.2012.

[7] Wikipedia, "Spiral model", http://en.wikipedia.org/wiki/Spiral_model, pristupljeno 21.06.2012.

[8] Wikipedia, "IBM Rational Unified Process", http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process, pristupljeno 22.06.2012.

[9] Wikipedia, "Test-Driven development", http://en.wikipedia.org/wiki/Test-driven_development , pristupljeno 22.06.2012.

[10] Wikipedia, "Agile software dovelopment", http://en.wikipedia.org/wiki/Agile_software_development, pristupljeno 22.06.2012.

[11] Wikipedia, "Scrum (development)", http://en.wikipedia.org/wiki/Scrum_(development), pristupljeno 23.06.2012.

[12] Wikipedia, "Lean Development", http://en.wikipedia.org/wiki/Lean_software_development, pristupljeno 23.06.2012.

[13] Wikipedia, "Extreme programming ", http://en.wikipedia.org/wiki/Extreme_programming, pristupljeno 23.06.2012.