UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miha Šinkovec

RAZVOJ APLIKACIJ Z METODOLOGIJO PROTOTIPIRANJA

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Marko Bajec

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01608/2009
Datum: 15.10.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MIHA ŠINKOVEC**

Naslov: **RAZVOJ APLIKACIJ Z METODOLOGIJO PROTOTIPIRANJA**
USING PROTOTYPING IN SOFTWARE DEVELOPMENT

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Pri razvoju informacijskih sistemov se zahteve stalno spreminjajo in če tega ne upoštevamo, se kmalu zgodi, da končni informacijski sistem ne ustreza aktualnim zahtevam. Eden možnih pristopov, ki se s tem sooča, je prototipiranje.

V okviru diplomskega dela preučite metodologijo prototipiranja in na praktičnem primeru prikažite njeno uporabnost.

Mentor:
prof. dr. Marko Bajec

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:
prof. dr. Franc Solina

IZJAVA O AVTORSTVU

Diploskega dela

Spodaj podpisani MIHA ŠINKOVEC,
z vpisno številko 63030057,

Sem avtor diplomskega dela z naslovom:

Razvoj aplikacij z metodologijo prototipiranja

S svojim podpisom zagotavljam da:

- sem diplomsko delo izdelal samostojno pod mentorstvom **izr. prof. dr. Marka Bajca,**
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 12. 04. 2010 Podpis avtorja:

Zahvala

Za strokovno pomoč in nasvete pri pisanju diplomske naloge se zahvaljujem izr. prof. Marku Bajcu in mlademu raziskovalcu Štefanu Furlanu.

Zahvaljujem se moji družini, ki me je tekom moje študijske poti podpirala.

Kazalo

Seznam slik

Seznam tabel

Seznam uporabljenih kratic in pojmov

XML *eXtensible Markup Language*	Razširljiv označevalni jezik.
HTML *HyperText Markup Language*	Označevalni jezik za oblikovanje spletnih dokumentov.
JS *JavaScript*	Skriptni jezik, ki omogoča programski dostop do objektov v odjemalcu. Integriran v brskalnik odjemalca.
AJAX *Asynchronous JavaScript and XML*	*Asinhroni JavaScript* in *XML.*
CSS *Cascading Style Sheet*	Stilne predloge za oblikovanje spletnih strani.
MVC *Model – View – Controller*	Tip arhitekture.
PHP *Hypertext Preprocessor*	Splošnonamenski skriptni jezik, tipično uporabljen pri razvoju spletnih aplikacij.

Povzetek

V današnjem času se poslovni sistemi spreminjajo hitreje, kot se obrne konvencionalni kaskadni življenjski cikel. Iz tega lahko sklepamo, da realizirani programski sistem ne bo več predstavljal odgovora na aktualne, v dobi razvoja že spremenjene uporabnikove zahteve. Tega problema ne uspemo izničiti niti s povečano storilnostjo oziroma produktivnostjo.

Kot ustrezna rešitev navedenega problema se je kot ena izmed variant pokazalo prototipiranje. Namesto, da gradimo in razvijamo celotni sistem, gradimo model, katerega lahko hitro in enostavno v procesu prilagajamo uporabnikovim zahtevam. Take modele oziroma prototipe lahko sproti predstavljamo uporabnikom, ti jih ocenijo in če niso zadovoljni z njimi, lahko tak model hitro in enostavno modificiramo oziroma zavržemo, kar pa ni pogojeno z velikimi stroški, kot bi to bilo v primeru spreminjanja že obstoječega sistema.

Zahteve so dinamične, hitre in cenovno racionalnejše.

V diplomski nalogi, v prvem delu, bom predstavil zgodovino in pregled klasičnih in novejših agilnih metodologij razvoja. Podrobno so predvsem ponazorjene njihove prednosti in slabosti, katere so se izkazale v preteklosti.

Skozi oči vodje projekta sem nato poskušal predstaviti njegove prednosti, težave ter orodja uporabljena pri razvojnem procesu s pomočjo prototipiranja. Na primeru razvoja spletne aplikacije pa tudi postopek uporabljen v praksi.

Ključne besede: življenjski cikel razvoja programske opreme, agilne metodologije, prototipiranje, razvoj spletne aplikacije.

Abstract

Today the business system changers faster than the usual conventional cascade life cycle. Because of that, we can conclude, that today's programming system will no longer be presented as the answer to this topic in the developing age of ever changing user requirements. Neither increased performance or higher productivity will decrease the problem.

The appropriate solution to this stated problem is prototyping. Instead of building and developing the whole system, we build a module that can be easily adjusted throughout the process of programming to the user's requirements. Such modules, or so called prototypes, can be presented to users during the programming process. The users can evaluate, and in the case of insufficiency, the module can be quickly and easily modified or abandoned. This process is unrelated to the big expenses, as it would be in the process of a presently conventional system. Requirements are dynamic, quick, and price rational.

In the first part of my thesis I review the history and evolution of classical and modern agile methodology. I illustrate in detail the advantages and disadvantages that have shown up in the past. Through the eyes of the project leader, I try to present their advantages, struggles, and the tools they used in the process of development utilizing the help of prototyping. I also present the example of development of web application that uses this process of prototyping.

Key words: system developement life cycle, agile methodology, prototyping, development of web application.

1 Uvod

V današnjem času računalniško podprt informacijski sistem v podjetju ni več izbira, ampak nuja. Velika količina podatkov, ter iz njih pridobljene informacije, so osnova odločanju podjetja. Podjetja, ki se ne uspejo hitro odzvati na nenehne spremembe poslovnega okolja, niso konkurenčna. Vprašanje, ki se na tem mestu poraja je: kako pravzaprav narediti dober računalniško podprt informacijski sistem? Pri razvoju bodočega sistema namreč sodelujeta dve skupini ljudi z različnimi znanji in interesi. Na eni strani so tisti, ki so tehnično sposobni narediti sistem, na drugi pa tisti, ki razumejo poslovno področje, ki ga mora sistem podpirati.

V preteklosti so se pojavile različne metodologije razvoja informacijskih sistemov. Njihov namen je bil v prvi vrsti formalizacija razvoja skozi celoten življenjski cikel razvoja. Prve metodologije so se zanašale na jasno definirane postopke, ki naj bi vodile delo razvijalcev sistema, tako da poskušajo analizirati čim večji del potreb kupca pred implementacijo funkcionalnosti v sistem. Takšen pristop se ni izkazal za uspešnega pri vseh podjetjih. Vse bolj jasno je postalo, da ustvarjanje preobsežnih količin dokumentov, ki opisujejo delovanje sistema, ne more zagotoviti dobrega informacijskega sistema v poslovnem okolju, kjer se spremembe nenehno odvijajo.

Agilni razvoj, kot ime za filozofijo v ozadju številnih novejših metodologij, poskuša odpraviti težave tradicionalnega razvoja sistemov.

Prototip, ki izhaja kot model iz te filozofije je dinamična rešitev za dinamične zahteve. Ta je skozi nenehne iteracije prilagodljiv, poudarek pa ni na postopkih dela, temveč na ljudeh kot posameznikih. Tudi prototipni razvoj ni univerzalna rešitev mnogih težav pri razvoju informacijskih sistemov. Pravzaprav deluje najbolje le tam, kjer je poslovno okolje turbulentno, specifikacije se nenehno spreminjajo, razvojne skupine pa so razmeroma majhne.

Ideja te diplomske naloge je, da lahko uporabimo prvine in metode prototipnega razvoja na primeru implementacije spletne aplikacije. Z njegovo uporabo skušamo doseči boljšo zadovoljitev kupca glede njegovih specifičnih poslovnih potreb, skušamo doseči večjo prilagodljivost in boljšo upravičenost stroškov.

Tretje poglavje opisuje osnovne pojme povezane s potrebo po metodologijah razvoja. Opredeli pojem metodologije ter prikaže najbolj znane družine metodologij razvoja.

Četrto poglavje predstavi agilni razvoj ter osnovne prvine. Nakaže težave tradicionalno vodenih projektov, ter ponazori kazalce pri odločanju o izbiri ustrezne metodologije.

Peto poglavje predstavi model prototipnega razvoja. Prikaže različne klasifikacije prototipov, ter vsako posamezno tudi opiše. Skozi orodja za razvoj prototipov poda potrebe po lastnostih, ki jih razvojna ekipa išče v posameznem orodju. Poda v katerih primerih je prototipiranje dobrodošlo, ter v katerih primerih se prototipiranje v celoti odsvetuje.

V šestem poglavju je predstavljeno ozadje aplikacije, katero bomo v prihodnjih korakih prikazali v praksi, ter ozadje o izboru prototipiranja.

V sedmem poglavju je uporaba modela prototipiranja uporabljena na primeru razvoja spletne aplikacije. Odločitev o posameznih korakih, orodjih, ter prikaz končnih rešitev.

2 Življenjski cikel programske opreme

Življenjski cikel aplikacije oziroma produkta, se prične z zasnovo in konča z vzdrževanjem pri uporabniku. Proces razvoja razčlenimo na zaporedje medsebojno odvisnih aktivnosti, ki temeljijo na potrebi po izdelavi uporabnega produkta. Termin »cikel« izhaja iz dejstva, da vsak razviti produkt generira nove potrebe.

Poznamo več različnih modelov, ki se razlikujejo po imenu in številu faz, s skupnim imenovalcem pa jih lahko predstavimo v naslednjih šest faz:

- Analiza
- Načrtovanje
- Implementacija
- Testiranje
- Inštalacija
- Obratovanje in vzdrževanje

Analiza

Vsak projekt je v splošnem rezultat zahtev, katere formuliramo v obliki dokumenta, ki opisuje uporabnikove zahteve. Tak dokument imenujemo definicija zahtev in predstavlja izhodišče za načrtovalce, ki pripravijo predloge za izvedbo. Namen in cilj je izvedba podrobnega opisa zunanjega obnašanja sistema. Opis zajema funkcije, ki jim je potrebno zadostiti: omejitve, povezave z okoljem ter vse povezane informacije, ki opisujejo velikost sistema, hitrost izvajanja, lastnosti in ostale pomembne karakteristike. Dokument, ki v tej fazi nastane imenujemo specifikacija zahtev.

Načrtovanje

Načrtovanje zajema aktivnosti od specifikacije do definiranja implementacije. V začetku si načrtovalci pomagajo s specifikacijami, na osnovi katerih izvajajo dekompozicijo notranjih funkcij sistema. Ta postopek lahko poteka v več nivojih in postopoma. Rezultat te faze je seznam hierarhične in večnivojske strukture v obliki modulov, podanih v načrtu izvedbe.

Implementacija

Faza implementacije zajema razvoj programske opreme, katera bo zadovoljevala vsem zahtevam. Implementacija programov se prične s kodiranjem modulov na osnovi specifikacij (iz predhodne faze), testiranjem le-teh in konča z integracijo ter končnim testiranjem produkta.

Testiranje

Ločimo testiranje enot oziroma modulov, integracijsko testiranje ter sistemsko testiranje. Pri testiranju enot, kot osnovnih gradnikov programskega sistema, se osredotočimo na del programa, da lažje ugotovimo in odstranimo napake. Kontroliramo tudi obnašanje modula glede na podane specifikacije, kar imenujemo tudi funkcionalno testiranje. Po izpeljanem testiranju vsakega modula, nastalega v fazi implementacije, integriramo oziroma povežemo in združimo skupino, samostojno že testiranih modulov, v enotno programsko strukturo ter jih testiramo (integracijsko testiranje ali tudi testiranje programskih komponent). Sledi sistemsko testiranje, kjer preverimo, če se celoten programski sistem, postavljen v določeno strojno okolje, obnaša ustrezno, oziroma adekvatno, podanim specifikacijam zahtev programske opreme.

Obratovanje in vzdrževanje

Vzdrževanje lahko opišemo kot neprestano iskanje napak in njihovo odstranjevanje ter razširitve (dodajanje novih zmožnosti). V sklopu te faze ločimo med adaptivnim, preventivnim in perfektivnim vzdrževanjem. Konvencionalni pristop k življenjskem ciklu temelji na tem, da v fazi analize opredelimo, kaj naj sistem dela, torej je poudarek na funkcijah, ki naj jih sistem zadovoljuje. Šele v fazi načrtovanja, ki prevzame specifikacije, nastale z odgovarjanjem na vprašanje KAJ, opišemo kako pridemo do zahtevane funkcionalnosti. Pri načrtovanju torej preidemo k opisu notranje strukture sistema. Bistvo konvencionalnega pristopa k specifikaciji zahtev, je ločiti zunanje obnašanje sistema od notranje sistemske strukture.

3 Metodologije razvoja programske opreme

Na razpolago imamo veliko različnih metodologij. V tem poglavju bom na kratko predstavil razvoj metodologij, nekatere tipične predstavnike standardnih in agilnih metodologij, njihov namen in sestavo.

3.1 Definicija metodologije

Metodologijo lahko na splošno opredelimo kot:

»*... skupek postopkov, tehnik, orodij in pripomočkov za dokumentiranje, ki koristijo razvijalcem sistema pri njihovem prizadevanju implementirati nov informacijski sistem. Metodologijo sestavljajo faze, ki so same sestavljene iz podfaz in ki vodijo razvijalce sistema pri izbiri tehnik, ki so primerne v vsaki fazi projekta, ter jim pomagajo načrtovati, upravljati, kontrolirati in ocenjevati projekte razvoja informacijskih sistemov.*« (Avison, Fitzgerald, 1996, str. 10).

Avtorja ugotavljata tudi, da je metodologija še več kot le skupek njenih sestavin. Metodologije se razlikujejo v posameznih tehnikah, ki jih priporočajo, o postopkih, ki jih predpisujejo ali v vsebini posamezne faze, toda včasih gre celo za bolj fundamentalne razlike.

3.2 Osnovni pojmi metodologije

Cockburn metodologije deli na trinajst osnovnih pojmov [17]:

- **Vloge**, ki jih prevzemajo in opravljajo zaposleni, določajo njihovo delo. Razvoj poteka na nivoju skupin, znotraj katerih je nujna potreba po različnih vlogah. Določitev zaposlenega glede na vlogo je treba oceniti glede na njegove osebne lastnosti in strokovno znanje. Vsaka vloga ima uveljavljene postopke, po katerih vsak zaposleni deluje. Vsaka vloga more vsebovati aktivnosti, ki so za posamezni projekt koristne, sicer se jo iz projekta odstrani.
- **Skupina:** je združba ljudi z različnimi vlogami, ki deluje na določenem projektu. Na posameznem projektu lahko deluje tudi več skupin, kar zahteva obsežnost projekta.
- **Postopki:** predstavljajo osnovo metodologije. Metodologija določa postopke, ki jih je potrebno storiti za dosego ciljev. Ti se navezujejo na osebe, vloge, skupine, način dela in izdelke.
- **Aktivnosti:** predstavljajo dela, ki jih zaposleni izvajajo. Navadno so določene z vlogami na projektu. Vsebina aktivnosti je določena z zaporedjem postopkov in časovnimi roki.
- **Orodja:** so pripomočki, katere uporabljamo pri razvoju. Pri nekaterih metodologijah uporabljamo preproste, pri drugih pa novejše. Njihov skupni cilj je omogočiti čim hitrejše delo.

- **Izdelek:** je to, kar poskušamo s pomočjo metodologije ustvariti. To je tudi končni cilj. Njegova gradnja poteka z aktivnostmi in postopki. Nanj predvsem vpliva kakovost izdelave in uporabljeni standardi.
- **Kakovost:** določa natančnosti izdelave pri aktivnostih. Določamo jo na podlagi rezultatov meritev izraženo predvsem s številom skritih napak in preprostostjo vzdrževanja.
- **Standardi:** vplivajo na delovanje in izgled izdelka, lahko so širše sprejeti, kot le dogovor med razvijalci. Vplivajo na delovanje in izgled izdelka.
- **Proces:** je točno določeno zaporedje izvajanja aktivnosti.
- **Mejniki:** z njimi določamo stanje projekta, saj z mejniki, katere ne dosežemo, pokažemo kaj je v okviru projekta še potrebno narediti. Nekateri so le dejstva, drugi vključujejo dejanje.
- **Strokovno znanje:** je eden od osnovnih kriterijev opravljanja določene vloge. Minimalno znanje, ki je potrebno pri razvoju programske opreme, je poznavanje programskega jezika in razvojnega okolja.
- **Skupinske vrednote:** so skupek splošno sprejetih vrednosti razvojne skupine, ki pomembno vplivajo na elemente metodologije. Razlikujejo se od skupine do skupine.
- **Osebnost:** je vzorec delovanja posameznika. Ujemanje osebnosti in njegove vloge omogoča boljše uspehe posameznika.

3.3 Koristi metodologije

Pri razvoju programske opreme, metodologija zagotavlja kvaliteten zajem naročnikovih zahtev in komunikacijo med naročnikom in razvijalci. To zagotavlja dober vpogled nad samim stanjem projekta.

Koristi metodologije se opazijo pri:

- **Vpeljavi novih ljudi v projekt**
  Ker je iz vpeljanega sistema razvidno, kam posameznik spada znotraj hierarhije ter kako poteka delo, se podjetja lažje in kvalitetnejše vključijo v sam razvoj.

- **Zamenjavi ljudi na projektu**
  Ker je hierarhija poznana, je zamenjava izvajalca lažje in hitreje izvedena.

- **Pregled stanja projekta**
  Zaradi stalnega pregleda nad opravljenim delom je vodenje projekta uspešnejše. Prav tako je pregled nad neizvedenimi nalogami enostavnejši. Bistvena prednost je ta, da je naročniku omogočen stalni pregled nad napredkom projekta.

- **Določitev delovnih mest pri projektu**
  Metodologija določa uporabo znanih tehnik in standardov, ki so potrjeni in preizkušeni. Zaposlenim se tako lahko s pomočjo dodatnih izobraževanj omogoči hitrejše in kvalitetnejše delo.

3.4 Pregled znanih metodologij

Poznamo več modelov metodologij, pri čemer vsak predstavlja neko zaporedje korakov, katerih se razvijalci držijo pri razvoju programske opreme. Metodologija pri tem vsebuje: iz modela sestavljene prakse in priporočila ter shemo delovanja in način izobraževanja.

Najpomembnejše in najbolj znane družine metodologij razvoja programske opreme so:

- Modeli tradicionalnih metodologij:
  - Model od zgoraj navzdol (angl. *top – down model*)
  - Model od spodaj navzgor (angl. *bottom – up model*)
  - Kaskadni oziroma Slapovni model (angl. *waterfall model*)
  - Inkrementalni model (angl. *incremental model*)
- Modeli evolucijske metodologije:
  - Spiralni model (angl. *spiral model*)
  - Prototipni model (angl. *spiral model*)
  - Ciklični in inkrementalni model (angl. *iterative and incremental model*)
- Agilne metodologije:
  - Ekstremno programiranje (angl. *extreme programming*)
  - *Scrum*
  - Vitek razvoj
  - Funkcijsko voden razvoj

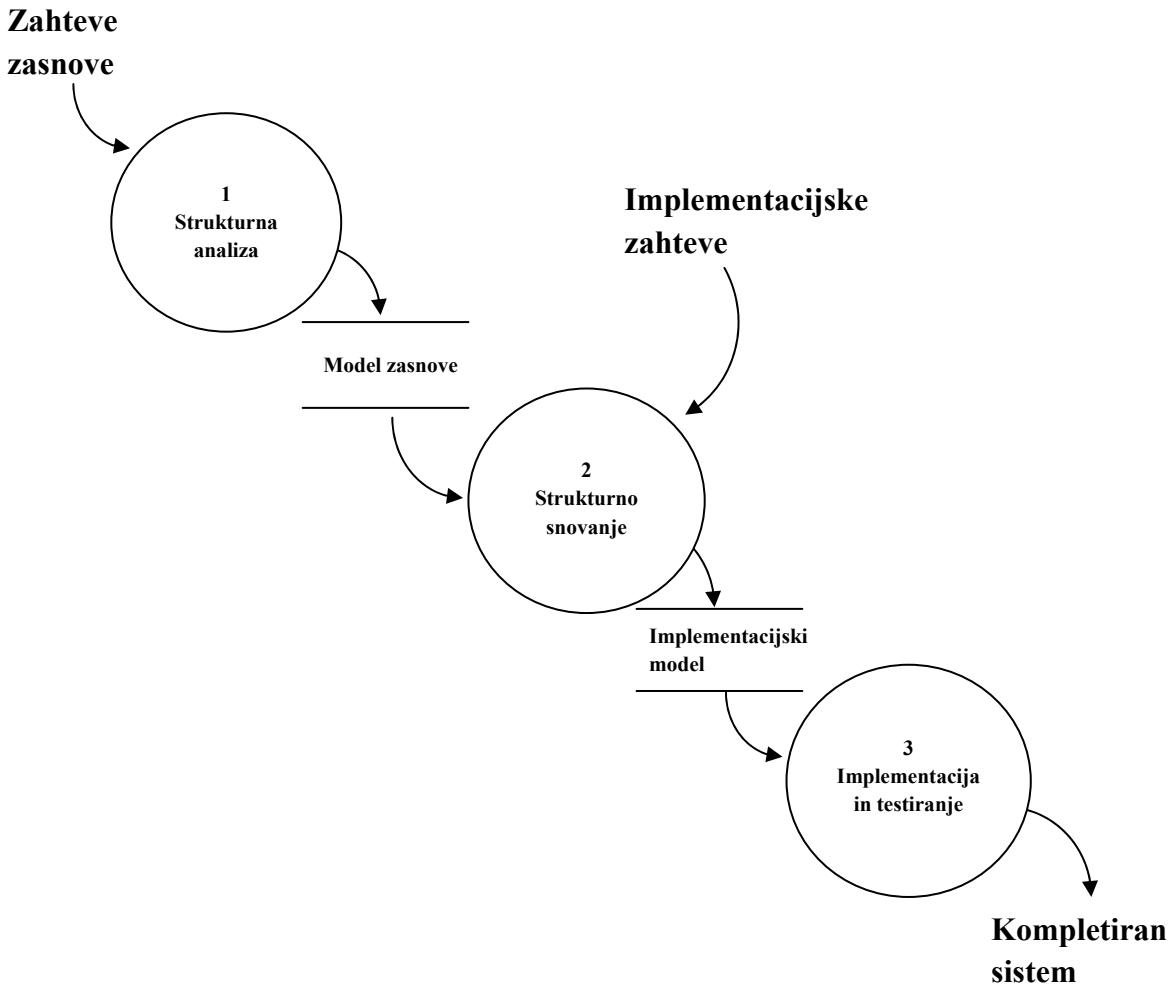
3.4.1 Modeli tradicionalnih metodologij

3.4.1.1 Model »od zgoraj navzdol« in »od spodaj navzgor«

To je družina tradicionalnih metodologij iz 60. in 70. let prejšnjega stoletja. Značilno za to obdobje je bilo, da so se takrat začeli izvajati prvi večji projekti, kar je botrovalo zahtevam za vzpostavitev razvoja pravil koordinacije in nadzora sistema. Pojavila sta se prva principa, razvoj »od zgoraj navzdol« in »od spodaj navzgor«.

3.4.1.2 Kaskadni ali sekvenčni model – Waterfall modell

Teksti, ki so zapisani v krogih, predstavljajo aktivnosti, ki imajo svoje vhodne in izhodne podatke (dokumente), ki so verižno povezani med sabo in na diagramu označeni z ustreznimi loki (puščicami) oziroma pravokotniki z imeni podatkov oziroma dokumentov, ki se prenašajo med aktivnostmi. Razvojni cikel se prične z zahtevami zasnove ali specifikacijo zahtev, ki jih poda naročnik in/ali vodja projekta. V splošnem se aktivnosti izvajajo v vrstnem redu, ki je podan na sliki, vendar je možen tudi drugačen vrstni red, med drugimi tudi interaktivno prehajanje (vračanje) med aktivnostmi ter preverjanje in morebitno modificiranje po vsaki izvedeni aktivnosti.



Slika 1: Kaskadni model življenjskega cikla programske opreme

3.4.1.2.1 Prednosti

Pokazalo se je, da je napaka, odkrita v zgodnejših fazah razvoja, cenejše in časovno učinkovitejše odpraviti. Ocenjuje se, da je odprava pomanjkljivosti tudi od 50 do 200 krat dražje odpraviti v fazi izdelave, kot če bi to storili že v fazi načrtovanja. V najslabšem primeru se nam lahko zgodi, da šele v fazi izdelave ugotovimo, da je neko komponento nemogoče izvesti.

Veliko jih tudi prisega na kaskadni model, zaradi njegove discipline in preprostosti uporabe, saj so postopki strukturirani, sam razvoj pa napreduje linearno, kar z lahkoto napoveduje napoved mejnikov v integraciji.

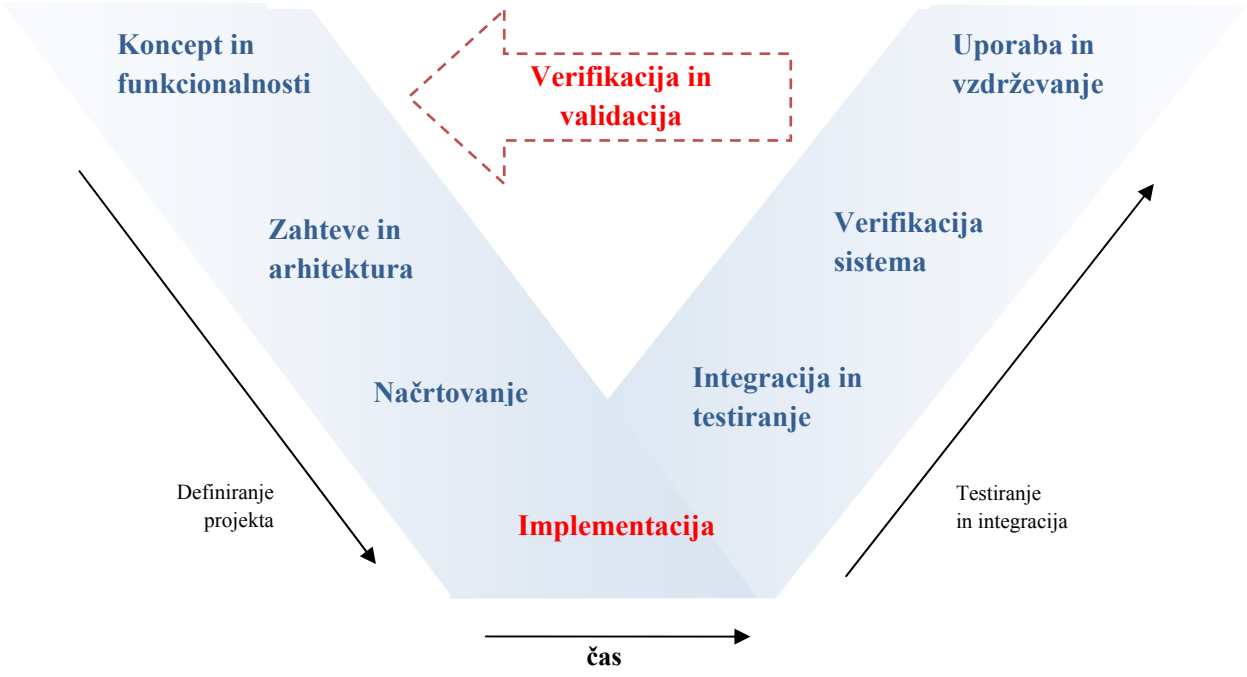
3.4.1.2.2 Slabosti

Kar naj bi bile njegove prednosti, so se za mnoge izkazale za slabosti v praksi. Namreč prepričanje, da bomo uspeli ugotoviti vse pomanjkljivosti v posamezni fazi življenjskega cilja so se izkazale za nemogoče.

Velikokrat se zgodi, da naročniki niso seznanjeni z vsemi zahtevami, ki jih bomo potrebovali v naslednjih fazah. Če naročnik spremeni svoje zahteve po tem, ko smo že v zaključni fazi načrtovanja, je potrebno načrt spremeniti, tako da se ga prilagodi novim zahtevam, kar privede do povečanja stroškov.

Prav tako se oblikovalci ne zavedajo težav, ki bodo nastale pri izvajanju v prihodnosti. Te postanejo jasne šele v izvedbeni fazi. Tako je bolje spremeniti načrt, kot pa vztrajati pri uporabi modela, ki je bil prvotno narejen na podlagi napačnih napovedi.

3.4.1.3 »V« ciklični model – V cycle



Slika 2: »*V*« model razvoja programske opreme

»*V*« model razvoja aplikacij, je nekoliko izpopolnjen kaskadni model. Pri »*V*« modelu imamo na levi strani aktivnosti verifikacije projekta, na desni strani pa so aktivnosti faze validacije.

V fazi *koncept funkcionalnosti* projektna ekipa skuša najprej od naročnika, s pomočjo anket in pregleda organizacije, zbrati kaj naročnik dejansko potrebuje. V tej fazi smo predvsem osredotočeni na definiranju, kako naj bi idealni sistem deloval, pri tem pa se v tej fazi še ne oziramo kako bomo to storili. Kot produkt te faze nastane model zasnove, ki

v prihodnjih fazah služi kot opora pri odločanju. Prav tako v tej fazi določimo potrebe, ki bodo zadoščale pri testiranju funkcionalnosti pred predajo naročniku.

V fazi *zahtev in arhitekture* nato, na podlagi modela zasnove, sistemski inženirji oblikujejo arhitekturo in funkcionalnosti modela. V primeru težav pri implementaciji funkcij, se ustrezno obvesti naročnika in v dogovoru z njim eventualno dopolni dokument modela zasnove. V tej fazi razvoja se izdela dokument specifikacij sistema, ki bo služil v fazi implementacije. V njem so definirane datotečne strukture, organizacija sistema ter menijske strukture. Prav tako so navedeni primeri poslovnih funkcij ter izgledi poročil in grafičnega vmesnika, za boljše razumevanje.

V fazi validacije (desni del »*V*« modela) se ekipa predvsem ukvarja s testiranjem aplikacije. Na tem mestu se izvajajo testi enot ter analiza napisane kode. Prav tako se v tej fazi pregleda upošteva standarde kodiranja ter učinkovitost kodiranja.

Zadnja faza testiranja je izpolnjevanje zahtevam uporabnika, ki so bile definirane v fazi *analize in načrtovanja sistema*. S potrditvenim preizkusom naročnik sistem sprejme ali zavrne.

3.4.1.3.1 Prednosti

Tako kot kaskadni model, je tudi *V* model enostaven za uporabo. V vsaki fazi imamo vnaprej definirane produkte. Zaradi prenosa faze razvoja v zgodnejše faze življenjskega cikla, omogoča večje možnosti za uspeh v primerjavi s kaskadnim modelom.

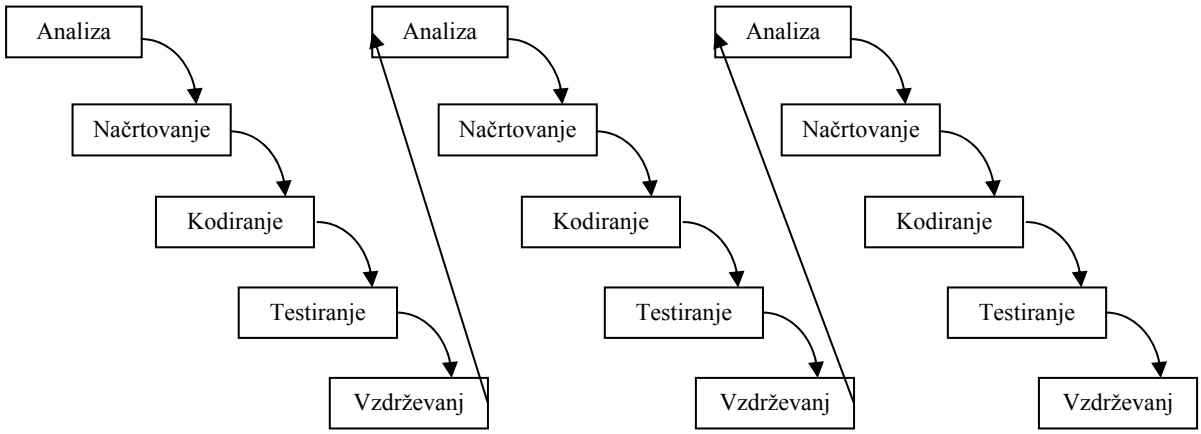
Uporaben je predvsem na manjših projektih, kjer so zahteve lahko razumljive.

3.4.1.3.2 Slabosti

Enostavnost uporabe *V* modela, pa prinaša njegovo togost pri uporabi. Prilagajanje spremembam je zaradi tega precej drago in oteženo. Slabost modela je tudi ta, da ne zagotavlja jasnih rešitev za težave najdene v fazi testiranja.

3.4.1.4 Inkrementalni model

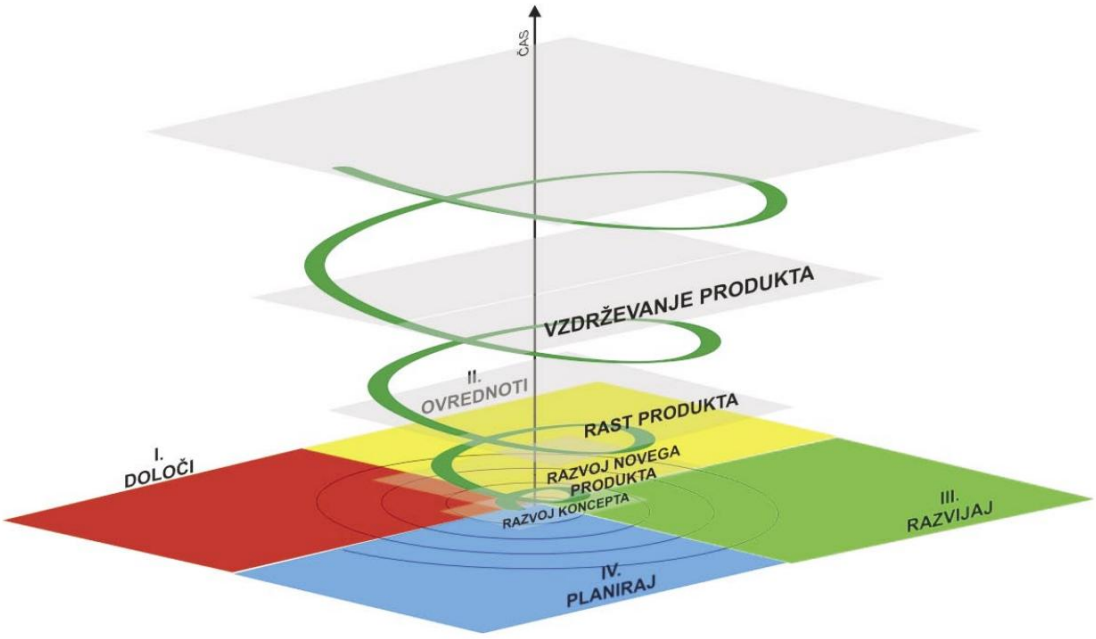
Cilj inkrementalnega razvoja je skrajšati čas razvoja. Pojavi se večje število korakov, v katerih se izvedejo vse faze razvoja *Slapovnega modela*. Razlika nastane v tem, da se razvoj vseh funkcionalnosti sedaj ne zgodi v enem koraku. Po končanju posameznega sklopa, se produkt preda naročniku v uporabo in naprej razvija do končnega sistema. Razvoj se prilagodi tako, da se najprej razvijejo ključni deli sistema [19].



Slika 3: Razvoj programske opreme s pomočjo inkrementalnega modela

3.4.2 Modeli evolucijskih metodologij

3.4.2.1 Spiralni model – Spiral modell



Slika 4: Spiralni model razvoja programske opreme

Spiralni model prevzema najboljše lastnosti zaporednega in prototipnega razvoja. Sestavljen je iz več ciklov, pri tem pa vsak cikel vsebuje vse glavne faze razvoja. V začetnih fazah so cikli prototipno usmerjeni, v kasnejših pa so cikli namenjeni nadgrajevanju sistema, odpravljanju napak ter vzdrževanju. V primeru, da naročnik ne poda zahteve dovolj jasno, se število ciklov do končne rešitve nekoliko poveča, pri tem pa se z vsakim obhodom bližamo rešitvi.

Korake v življenjskem ciklu lahko razdelimo v naslednje faze:

1) V začetnih fazah je potrebno, da so sistemske zahteve opredeljene čimbolj natančno. To običajno vključuje razgovor z večjim številom uporabnikov.

2) Kreiranje idejnega projekta za nov sistem. Ta del je najpomembnejši v spiralnem modelu. Vse možne alternative so analizirane, tako se določi stroškovno učinkovita strategija razvoja. V primeru, da obstajajo tveganja in negotovosti glede zahtev, se izdela prototip in tako ugotovi možne rešitve za reševanje v prihodnjih fazah.

3) Prvi prototip novega sistema je kreiran iz predhodnega modela. To je ponavadi okrnjena izvedba sistema s približnimi značilnostmi končnega sistema.

4) Naslednji prototipi so razviti z naslednjim postopkom:
   a. Ocenitev predhodnega prototipa s stališča njegovih prednosti, slabosti in tveganj
   b. Opredeli se zahteve novega prototipa
   c. Načrt in oblikovanje bodočega prototipa
   d. Izgradnja in testiranje bodočega prototipa

Spiralni model se v večini primerov uporablja pri razvoju iger, to pa predvsem zaradi velikosti projekta in nenehnega spreminjanja ciljev.

3.4.2.1.1 Prednosti

Model je precej bolje prilagojen na spremembe v vseh fazah življenjskega cilka in se na njih zato tudi lažje odzove. Razvijalci programske opreme so vključeni v razvoj že od samega začetka procesa.

Metodo spiralnega razvoja se uporablja predvsem pri velikih projektih. Za majhne projekte se je, kot alternativa, pojavila metodologija agilnega razvoja.

3.4.2.1.2 Slabosti

Njegove prednosti pa se izkažejo tudi za slabosti. Visoka mera prilagoditve določenemu projektu omejuje ponovno uporabo modela, saj prihaja do razlik pri razvojih različnih aplikacij.

Zaradi sprememb in nejasnosti v fazi razvoja, prihaja do tveganj povezanih z izpolnjevanjem zahtev proračuna in rokov izdelave.

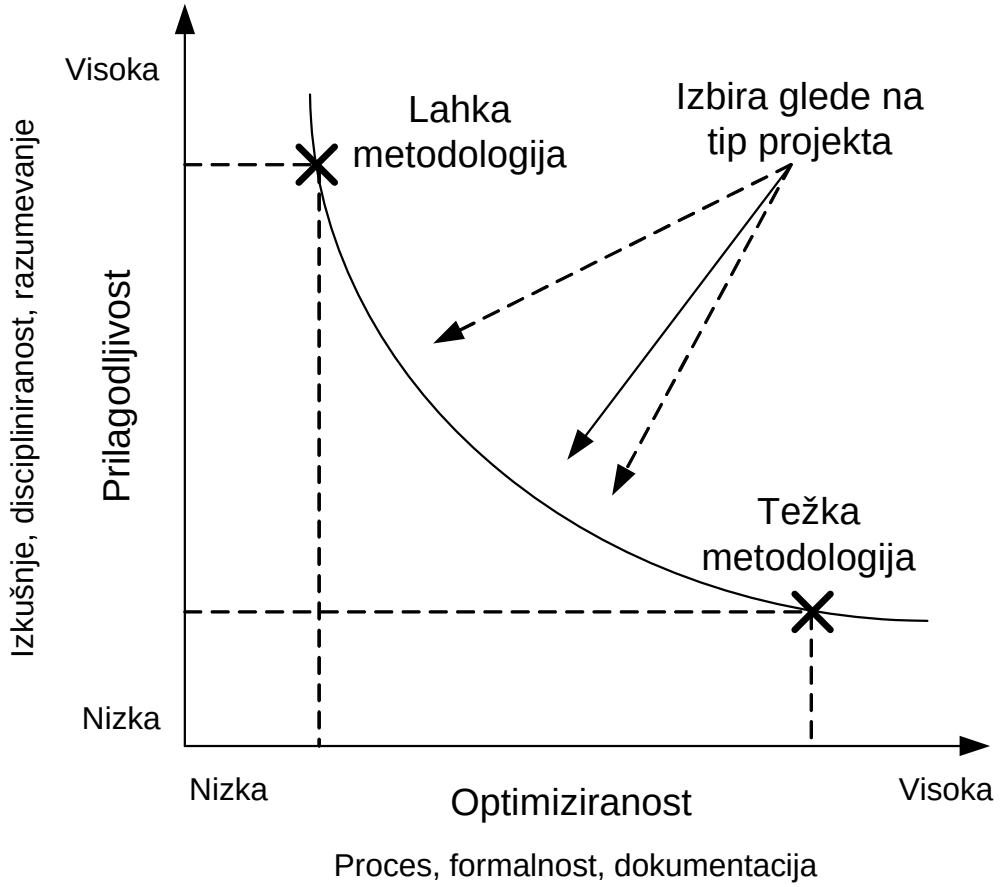
3.5 Vzdrževanje metodologije

Od sprejetja metodologije v podjetje, je potrebno metodologijo tudi vzdrževati. To je delo skrbnika metodologije. Ta skrbi za vse aktivnosti povezane z uporabo le te. Potrebno je poskrbeti, da se na začetku vsakega projekta določi primernost posamezne metodologije, ter se jo ustrezno prilagodi lastnostim projekta in podjetja. Med potekom projekta pa skrbi, da se vse osebe, vključene v projekt, držijo aktivnosti in jim je na razpolago v primeru morebitnih težav.

Po koncu projekta izdela ustrezno analizo, s pomočjo katere ustrezno dopolni in popravi postopke metodologije, seveda, če je to potrebno [20].

4 Agilne metodologije

Pri integraciji aplikacij gre za mehanizem sklapljanja posameznih aplikacij v enovito rešitev. Integracija aplikacij poskrbi tudi za enovit vmesnik k obstoječim sistemom. Poleg tega razširja funkcionalnost obstoječim sistemom, ne da bi jih bilo potrebno zamenjati z novimi sistemi. Tako lahko obstoječi sistemi nadaljujejo s svojim delom, hkrati pa lahko dodamo nove tehnologije, ki popeljejo organizacijo na višji nivo.



Slika 4: Agilna izbira metodologije

4.1 Težave pri tradicionalno vodenih projektih integracije

Tradicionalne metode so osnovane na predpostavki, da rigorozno vnaprejšnje planiranje dokončno odpravlja negotovnosti pri izvajanju. Kot rezultat tega se mnogi projekti utopijo v t.i. »*paralizi analize*« in ustvarijo ogromne zahteve ter načrte zato, da lahko tekom projekta ugotovijo, kako so spreminjajoče se zahteve in nove ugotovitve vplivale na natančno izdelan plan.

Druga izmed pomanjkljivosti tradicionalnega pristopa je ta, da je zelo težko ugotoviti ali je v času trajanja projekta prišlo do ustreznega napredka in še pomembneje, ali stvari potekajo dobro.

Naslednja težava je zadovoljstvo poslovnih strank in končnih uporabnikov, saj obstaja veliko tveganje, da po dolgotrajnem izvajanju integracijskega projekta, le ta ne bo ustrezal zahtevam organizacije ali uporabnikov. Pojavlja se tudi nagnjenje k preobsežnem vnaprejšnjemu planiranju informacijske strukture, saj se ljudje nagibajo k razmišljanju o stvareh, ki bi utegnile biti potrebne nekje v prihodnosti. Rezultat je (pre)zapletena infrastruktura, ki je velikokrat lahko v neskladju z dejanskimi potrebami aplikacij, ki jih bomo integrirali.

4.2 Filozofija agilnega razvoja

Spoznanje, da v poslovnih procesih ni možno preprečiti sprememb in negotovosti, je privedlo do prilagoditve razvojnega procesa aplikacij tako, da je le ta pripravljen na spremembe v negotovosti. Bistvo agilnih procesov je ideja o preprostosti – nikoli ne proizvedi več, kot je potrebno in nikoli ne izdeluj dokumentov, ki poizkušajo predvideti prihodnost. Z večanjem količine dokumentov se povečuje napor in delo, ki ga moramo vložiti, da najdemo določeno informacijo ter da informacije obdržimo ažurne.

Čeprav temeljijo na zelo discipliniranih procesih so agilne metode v bistvu prilagodljive. To omogoča razvojnim skupinam, da vključijo spreminjajoče se poslovne in uporabniške zahteve.

V začetku leta 2001 je skupina *Agile Alliance* izdelala osnutek vrednosti in principov, ki razvojnim skupinam omogočajo hiter odziv na spremembe. Osnovne ideje filozofije agilnega razvoja programske opreme se kažejo v štirih vrednotah:

- Posamezniki in interakcije pred procesi in orodji,
- Delujoč program pred obsežno dokumentacijo,
- Sodelovanje s strankami pred pogajanjem o pogodbi,
- Odziv na spremembe pred sledenjem izdelanemu planu.

Podrobneje o posamezni ideji v nadaljevanju.

Posamezniki in interakcije pred procesi in orodji

S poudarkom na komunikaciji in povezanosti med razvijalci, se izboljšuje delovno okolje, motivacija in občutek pripadnosti. Če za projektom ne stojijo sposobni ljudje, tudi odličen razvojni proces ne more rešiti projekta. Sposoben razvijalec ni samo strokovnjak na svojem področju, razvite mora imeti tudi komunikacijske sposobnosti, sposobnosti skupinskega dela in mora biti prilagodljiv spremembam v okolju.

Delujoč program pred obsežno dokumentacijo

Pri programiranju se stremi k preprostosti, ki temelji na naprednih rešitvah. Nove delujoče in preverjene verzije programa se izdajajo v rednih intervali. V projektih, kjer se

stremi na pretirano dokumentiranje, nastanejo težave, saj pisanje dokumentacije ovira razvoj. Največje težave nastajajo pri potrebi po usklajevanju in ažuriranju dokumentacije. Zato se na tem mestu odločimo le za dokumente, ki so nujno potrebni, ter ostale prioritete posvetimo razvoju kode.

Sodelovanje s strankami pred pogajanjem o pogodbi

Podrobne pogodbe nimajo prednosti pred komunikacijo in povezavo med naročnikom in razvijalci. Pogodbe so osnova, vendar je v njih poudarjeno sodelovanje in komunikacija med obema stranema.

Uporabnik najbolje ve kaj potrebuje in lahko le s kvalitetno komunikacijo to preda izvajalcu, ki tako lažje zagotovi, da je končni izdelek po meri naročnika.

Odziv na spremembe pred sledenjem izdelanemu planu

V primeru sprememb med samim razvojem projekta, je razvojna skupina pooblaščena za spreminjanje ciljev. Pogodba mora zato biti napisana tako, da omogoča naknadno spreminjanje ciljev projekta na podlagi komunikacije z naročnikom. Ni namreč idealna rešitev sledenje stalnicam in planu v primeru poslovnih sprememb.

4.3 Pomembni postopki agilnega projekta integracije

Cilj projekta integracije je zagotoviti kvalitetno integrirano rešitev, ki bo sprejemljiva za stranko in končana skladno z načrtom in dodeljenimi finančnimi sredstvi. Dobro voden projekt integracije, zasnovan na predlaganem procesu integracije, zagotavlja osnovo za planiranje, implementacijo in dokončanje integracije.

Postopki, ki jih moramo upoštevati pri agilnem projektu integracije so:

- **Iterativen razvoj** – razvoj v mnogih krajših iteracijah. Hitra integracija omejenega nabora aplikacij omogoča uporabnikom, da vidijo del končne rešitve in nudi povratne informacije skozi razvojni cikel. To znatno zmanjšuje tveganje, da bi izpustili ključne funkcije ali integrirali funkcije, ki jih ne potrebujemo. Hkrati omogoča uporabnikom, da se hitreje zavedajo prednosti, ki jih integriran sistem prinaša.

- **Inkrementalni razvoj** – kljub temu, da integracijo razdelimo na več manjših nalog, je malo verjetno, da bomo vsako nalogo zadovoljivo rešili že v prvem poizkusu. Ponavadi se učimo iz lastnih napak, ki jih v naslednjih inkrementih poizkušamo odpraviti.

- **Poenostavitev pri načrtovanju in razvoju** – koncept poenostavitve je tesno povezan z iterativnim razvojem. Raje, kot da pol leta razvijamo ogrodje, rešimo točno določen problem v nekaj tednih in s tem pokažemo, da razumemo kakšne so poslovne potrebe. Seveda poenostavitev ne pomeni, da je rešitev lahko površna, temveč da se problema lotimo direktno, ne da bi vnaprej izgradili ogromno infrastrukturo za podporo majhnemu delu funkcionalnosti. Z gradnjo infrastrukture skupaj s funkcionalnostjo zagotovimo, da infrastruktura podpira to, kar je za poslovanje resnično potrebno.

- **Prototipiranje** – s pomočjo prototipov odgovorimo na izzive integracije in ocenimo ali je možno izdelati rešitev na izbran način in z izbrano tehnologijo. Ponavadi ga uporabimo za ocenitev, preverjanje in potrditev izbranih arhitektur ter rešitev za integracijo, pogosto pa tudi za preverjanje zahtev, zmogljivosti in razsežnosti. Več o prototipiranju v nadaljevanju.

- **Ponovna uporaba** – pomeni zmožnost, da razvijemo nove aplikacije s pomočjo obstoječih rešitev. Ker večina obstoječih sistemov ni bila razvita z mislijo na ponovno uporabo, je potrebno poiskati rešitve, ki bodo naredile obstoječe sisteme ponovno uporabne. V mislih moremo imeti tudi ponovno uporabo na višjem nivoju abstrakcije – ponovno uporabo idej in dobrih rešitev v obliki ponovne uporabe vzorcev.

5 Prototipiranje

Prototipiranje je tehnika konstruiranja delne implementacije sistema, tako, da se lahko uporabniki in razvijalci podrobneje seznanijo s problemom oziroma njegovo rešitvijo.

Kot vidimo, smo uporabili izraz delna implementacija, to pa zato, ker če bi bila popolna, bi to že bil sistem in ne le prototip.

Poznamo več možnih klasifikacij prototipov:

- **Raziskovalni prototipi** – ugotavljamo želje uporabnika od sistema, torej, kaj naj bi grajeni programski sistem zagotavljal. Takšne prototipe pa nato zavržemo.
- **Eksperimentalni prototipi** – služijo kot pripomoček za ugotavljanje izvedljivosti (realizacije); nato jih zavržemo.
- **Razvojni prototipi** – razvijamo jih postopoma in z namenom, da v določenem času postanejo sam sistem oz. del sistema.

Kot vidimo sta dve najpogostejši delitvi znotraj prejšnje klasifikacije:

- Metoda zavračanja (*throwaway*)
- Razvojna (*evolutionary*) metoda, kot sistematični pristop pri izgradnji

Medtem, ko pri metodi zavračanja po pridobitvi vsega potrebnega znanja o problematiki in rešitvah prototip zavržemo in začnemo za tem razvijati sistem pri razvojni metodi prototip preraste v sam sistem.

5.1 Delitev metod prototipiranja

5.1.1 Metoda zavračanja prototipov

Pri gradnji zapletenih sistemov so zahteve ob samem začetku projekta praviloma nejasne in slabo definirane, včasih pa uporabnik sploh ne ve, kaj od končnega produkta pričakuje. Tako se zahteve med samo izgradnjo sistema velikokrat spreminjajo. Ponavadi se resnične zahteve pojavijo šele takrat, ko lahko uporabniki sistem praktično preverijo.

Po metodi zavračanja zgradimo hiter in robusten (*quick-and-dirty*) prototip, ter ga predstavimo potencialnim uporabnikom ali naročnikom z namenom, da skupaj z njimi:

- Dosežemo izvedljivost želja (zahtev)
- Potrdimo nujnost posameznih funkcij
- Odkrijemo manjkajoče, torej zahteve, ki niso bile podane predhodno
- Raziščemo možnosti razvoja ustreznega uporabniškega vmesnika

S pomočjo pri pregledu pridobljenih podatkov, lahko z večjo verjetnostjo trdimo, da gradimo ustrezni sistem ter dokončamo specifikacije programskih zahtev. Prototipa od te

faze dalje seveda ne gradimo, če lahko identificiramo vse zahteve in se naročniki z njimi strinjajo. Hiter in robusten prototip lahko zgradimo že v fazi načrtovanja (tako preliminarnega kot podrobnega). Z njimi lahko validiramo strukturo programskega sistema, torej ugotovimo, ali zasnovana programska struktura zadovoljuje vsem zahtevam oziroma bo lahko tem zadovoljila v prihodnje.

Tukaj velja poudariti, da pri načrtovanju velja podobno kot pri analizi, da prototipa ne gradimo, če smo lahko dovolj učinkoviti tudi brez njega oziroma obstaja splošno soglasje glede ustrezne alternative. V fazi podrobnega načrtovanja, pa nam lahko hiter in robusten prototip pomaga pri vrednotenju določenega algoritma.

Vsekakor v vseh fazah razvoja (analizi, preliminarnem in podrobnem načrtovanju) ne smemo pozabiti, da gradimo hiter in robusten prototip. Pri tem, s pojmom *hiter*, mislimo predvsem na hitrost izgradnje prototipa, zato pri tem ni potrebno izgubljati preveč časa. Le tako lahko rezultate, oziroma ugotovitve, upoštevamo še pravočasno in jih koristno uporabimo. Pod pojmom *robusten,* pa govorimo v smislu kvalitete, saj ga tako ali tako nameravamo kasneje zavreči. Zato tukaj ni večjega poudarka na načrtovanju, komentiranju in testiranju. [7,8]

Koraki uporabljeni v postopku prototipiranja so:

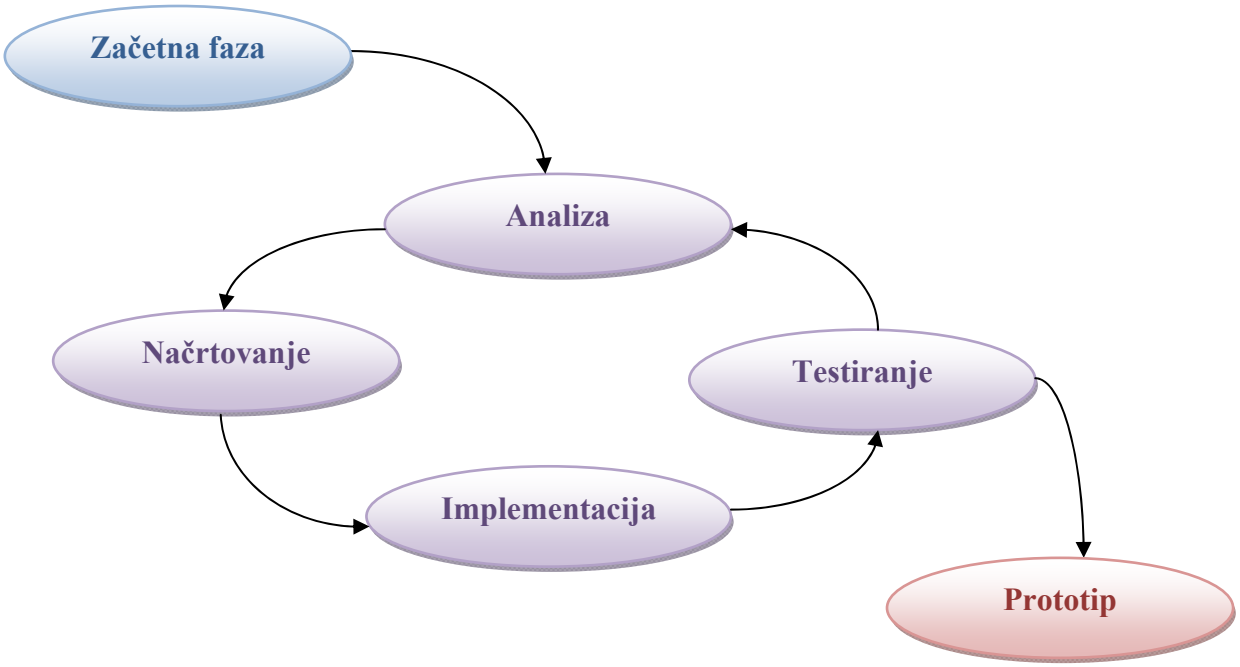
- Definiranje zahtev,
- Pregled uporabniških izkušenj, testiranje prototipa, ter določanje novih zahtev,
- Ponovitev (če je potrebno),
- Zapis končnih zahtev,
- Zavrnitev prototipa in izgradnja končnega sistema.

5.1.2 Razvojna metoda prototipiranja

Razvojna metoda prototipiranja se bistveno razlikuje od metode zavračanja prototipov. Bistvena sprememba je ta, da se pri tej metodi osredotočimo na izgradnjo zelo robustnega prototipa, ki ga konstantno dopolnjujemo, ob koncu pa bo predstavljal osrednji del novega sistema.

Zato pri tej metodi ne gledamo na hitrost, saj mora prototip imeti kvalitete končnega produkta, kot so zanesljivost ter enostavno vzdrževanje. Vse to povzroča počasnejši razvoj prototipa v primerjavi z metodo zavračanja.

V primeru izbire te metode, je naravna pot seveda ta, da sledimo običajnemu razvoju življenjskega cikla.



Slika 6: Faze razvojne metode prototipiranja

Potrebno je razjasniti delovanje končnega izdelka, priskrbeti uporabniške izkušnje na osnovi pridobljenih izkušenj, ponovno opraviti povpraševanje, ponovno načrtovati, kodirati in testirati. Proces seveda ponavljamo tako dolgo, dokler ne dosežemo zadovoljive stopnje prototipa, katerega razvijemo v končni sistem.

Tak način nam zagotavlja potreben pregled nad sistemom in potrebne dokumentacije.

5.1.3 Sistem delnega prototipiranja

Prototip lahko predstavlja delovanje celotnega sistema, poznamo pa tudi take, ki se ukvarjajo zgolj samo z določenim delom prototipa.

Te delne prototipe lahko zasledimo v naslednjih oblikah:

- **Pogovorni prototip** – dialog končnega sistema z uporabnikom. Prototip tega tipa simulira delovanje oziroma komunikacijo končnega sistema z uporabnikom. Je najpogostejša oblika delnega prototipa. Z njim dobijo uporabniki občutek komunikacije s sistemom, zato je ta vrsta prototipa zaželena, saj se lahko uporabniki že pred fazo implementacije seznanijo z delovanjem prototipa in predlagajo omejitve, izboljšave itd. Pogovorni prototip ima največji vpliv na prikaz uporabnosti in uporabnikovo dojemanje sistema, saj večina sistemskih analitikov in programerjev ni poučenih, kaj predstavlja za naročnika učinkovit dialog. Pogosto se oblikuje dialoge, ki niso dovolj jasni in/ali so zapleteni ter

zmedejo uporabnika. Zaradi vseh naštetih razlogov je priporočljivo izdelati prototip, ki ga lahko uporabniki testirajo in ocenijo, tako dialog izboljšamo uporabniku, še preden se lotimo izdelave dejanskega sistema.

- **Vnos podatkov** – Uporaben je predvsem pri sistemih, ki zahtevajo vnos velikega števila podatkov. Lahko se obravnava kot samostojna celota in ga kasneje povežemo z obstoječim sistemom. Izdelamo ga z namenom ugotavljanja hitrosti in točnosti vstavljanja podatkov. Z njim lahko tudi preverimo veljavnost in integriteto podatkov.

- **Sistem sporočanja** – Sporočila namenjena uporabnikom, je priporočljivo pred samo implementacijo sistema preveriti ter raziskati najprimernejši način njihovega podajanja.

- **Podatkovni sistem** – Prototip podatkovne baze lahko predstavimo kot nekaj zapisov, nato pa uporabniki in analitiki testirajo njihovo delovanje in identificirajo pomanjkljivosti. S takim prototipom je možno ugotoviti, katera polja so odvečna in katera manjkajo. Na tak način privedemo podatkovno bazo do optimalnega stanja, kar povzroči hitrejšo in lažjo obdelavo podatkov. Kreiramo lahko tudi takšne prototipe, ki uporabnikom in analitikom omogočajo kreiranje novih podatkovnih zapisov, opravljanje operacij nad njimi in prikaz le teh.

V povezavi s prototipi raziščemo še področje podatkovne baze pri prototipiranju. Podatkovna baza, uporabljena pri prototipiranju, je lahko dejanska ali le model dejanske podatkovne baze. V nekaterih primerih je dovolj, če sami modeliramo podatkovno bazo in razvijamo prototip, obstajajo pa tudi primeri, ko prototip gradimo na obstoječi podatkovni bazi. V tem primeru je potrebno biti pazljiv, saj te podatkovne baze ne smemo spreminjati, lahko pa jo pregledujemo.

Če smo naredili kopijo podatkovne baze in ta kopija nima nobene povezave z obstoječim sistemom, lahko tako bazo tudi spreminjamo. Ta način je seveda boljši, saj je bolj varen.

Pri prototipih te vrste obstajajo uporabniške zahteve do analitikov, da poiščejo različne vrste podatkov. Ugotoviti morejo kje se nahajajo ti podatki in kako do njih dostopati.

- **Računanje in logika** – Obstajajo primeri, ko so izračuni ali logika programskega sistema zelo zapleteni. Uporabniki v takem primeru podajajo primere pričakovanih izračunov oz. delovanja sistema. Te primere lahko nato priključimo sistemu ali jih povežemo z aplikacijo, s podatkovno bazo itd. Seveda nato primerjamo točnost prototipnih izračunov s podanimi.

- **Aplikacijski paketi** – preverimo uporabnost, da se izognemo slabemu povezovanju z drugimi aplikacijami. S prototipom preverimo majhne skupine

uporabnosti, če zadostujejo podanim pogojem. S tem se izognemo slabemu povezovanju z drugimi aplikacijami.

- **Koncept** – Včasih se pojavi vprašanje koncepta sistema. Zato je potrebno testiranje, da po nepotrebnem ne vlagamo denarja v razvoj velikih sistemov. Taki prototipi so razviti predvsem po metodi zavračanja saj morajo biti sestavljeni hitro in robustno, s tem lahko izluščimo pravi koncept sistema. Ko je koncept določen, se lahko lotimo načrtovanja ciljnega sistema.

5.1.4 Ekstremno prototipiranje

Ekstremno prototipiranje, kot proces razvoja, je predvsem uporabljen pri razvoju spletnih aplikacij. V osnovi se razvoj razdeli v tri razvojne faze, vsaka zasnovana na njeni predhodnici.

Prvo fazo sestavlja statični prototip, predvsem iz *HTML* strani. V drugi fazi opremimo zaslonske maske s simulacijskim slojem. V tretji fazi pa funkcionalnosti tudi dejansko implementiramo v prototip.

S procesom ekstremnega prototipiranja želimo opozoriti na drugo fazo razvoja, kjer je popoln funkcionalni grafični vmesnik zgrajen z zelo malo poudarka na storitve, ki niso del dogovora.

5.2 Orodja za prototipiranje

Potrebno je poudariti, da se je prototipiranje začelo uveljavljati šele po letu 1980. Ker metoda prototipiranja ni bila dodelana in je bila v začetnih fazah cena prototipiranja skoraj enaka ceni izdelave pravega sistema. V 80. letih pa so se pojavili jeziki četrte generacije, ki so na eni strani omogočali cenejše snovanje prototipov, po drugi strani pa so omogočili tudi njihovo hitrejšo in učinkovitejšo izdelavo. Ti jeziki nam omogočajo izgradnjo osnovnih prototipov v nekaj dneh, tako da uporabniku hitro prikažemo funkcionalnost bodočega sistema. Na pripombe in opozorila uporabnika lahko zato hitro odreagiramo ter prototip popravimo ali odpravimo pomanjkljivosti v dnevu ali dveh. Podatkovno bazo lahko hitro kreiramo in modificiramo.

Seveda jeziki četrte generacije niso namenjeni samo za generiranje prototipov, uporabimo jih lahko tudi za izgradnjo sistema. Vsekakor lahko prototipe v s pomočjo uporabe jezikov četrte generacije razvijemo v končni sistem.

Na tem mestu je potrebno poudariti, da izbira pravilnega orodja v veliki meri zavisi od izbrane metode prototipiranja (razvojna ali metoda zavračanja). V primeru odločitve za razvojno metodo, je smiselno razvijati prototip na istem okolju, kot bomo razvijali končni sistem. Jasno je, da je potrebno pri izbiri ustreznega orodja za prototipiranje upoštevati

tudi strojno opremo in ciljno okolje, v katerem bo deloval grajeni sistem. S stališča prototipiranja je potrebno izbrati tako orodje, da bo izdelava oziroma izgradnja prototipov čim hitrejša in enostavnejša, hkrati pa bo mogoče prototip brez večjega truda tudi spreminjati.

Optimalno orodje za prototipiranje je tisto, ki nam omogoča hitro in enostavno zajemanje uporabnikovih zahtev oziroma želja, ter jih učinkovito pretvoriti v končni sistem oziroma prototip. Prototipe gradimo z namenom omogočiti skupno delo oziroma sodelovanje uporabnikov in razvijalcev sistema. Nekatera orodja omogočajo sestavo prototipa med samim trajanjem delovnega sestanka uporabnikov in razvijalcev. Tak način dela je zelo zaželen, seveda pa je potrebno paziti, saj nekatera orodja, ki jih na tržišču prodajajo kot orodja za prototipiranje, še zdaleč niso to, za kar jih propagirajo.

5.2.1 Orodja za razvoj

Dobro orodje za razvoj prototipov naj bi :

- Bilo interaktivno
- Enostavno za uporabo
- Omogočalo enostavno in hitro spreminjanje
- Omogočalo hitro izgradnjo prototipov
- Vzpodbujalo postopno dograjevanje
- Podpiralo ustrezne strukture baz podatkov, ter vključevalo:
    - zmogljiv zaslonski oblikovalnik,
    - možnost povezovanja zaslonov kot odgovora na dialog,
    - zmogljiv generator poročil,
    - jezik četrte generacije ali generator kode,
    - primeren sistem za upravljanje baze podatkov,
    - integrirani podatkovni slovar,
    - možnost ekstrakcije podatkov iz obstoječih zbirk oziroma podatkovnih baz,
    - in njihov prenos v prototipno bazo podatkov (ali on-line dostop).

Pri razvojni metodi mora orodje dodatno nuditi še:

- možnost izboljšanja strojnih zmogljivosti,
- podporo podatkovnim strukturam končnega sistema, tudi glede distribuiranih podatkov,
- mrežni dostop,
- večuporabniško opcijo,
- sposobnost nadzora obsežnih baz podatkov,
- možnost vključevanja segmentov, napisanih v drugih programskih jezikih.

5.2.2 Orodja za simulacijo

Orodja za simulacijo omogočajo uporabnikom, da lahko na hitro zgradijo lahke, z animacijo opremljene simulacije ali ostale dele programske kode, ne da bi pri tem pisali kodo. Tako tehnični ekipi kot ostalim netehnično podkovanim članom v ekipi, omogoča pridobivanje uporabniških izkušenj, testiranje ter validiranje bodočega sistema. Priročno je to, da jim ta orodja generirajo tudi poročila, obvestila, zajeme zaslona in semantike.

Tu gre predvsem za nizko tvegano delo, vendar omejeno na besedila ali risanje na osnovi makete. Tako se, časovno zamudnega in z visokim tveganjem dela izgradnje prototipa preden bi nam ga naročnik šele potrdil, izognemo. Pri tem tveganje in stroške povezane z izvedbo programske opreme drastično zmanjšamo [17].

Pri simulaciji bodoče aplikacije lahko uporabimo tudi opremo, ki simulira v realnem času.

Nekatere od vodilnih orodij v tej kategoriji so *Irise, ProtoShare, Axure, Justinmind Prototyper* in *DefineIt.*

5.3 V katerih primerih uporabiti prototipiranje

Kot smo že omenili, pri razvoju programskih sistemov uporaba prototipov ni vedno primerna. Npr. pri paketni obdelavi prototipiranje sigurno ni tako učinkovito kot je to pri interaktivnih programih [7].

Prototipiranje je priporočljivo predvsem v naslednjih primerih:

- uporabniki ne vedo natančno, kaj želijo,
- uporabniki s težavo podajajo oziroma izražajo svoje zahteve,
- sistem spremeni osnove poslovnih operacij,
- uporabniški vmesnik je potrebno prilagoditi končnim uporabnikom,
- poslovne funkcije, ki naj bi jih računalniško podprli, so zapletene in nejasne ter jih uporabniki poznajo oziroma bolje razumejo kot analitiki,
- potrebno je preveriti zaslonske maske in poročila, iz njih pa ugotoviti možne izboljšave ter zvišati nivo uporabnosti,
- uporabniki ne razumejo in opazijo vseh pridobitev, ki jih bo nudil nastajajoči sistem,
- potreba po raziskavi vrednosti alternativnih rešitev.

Vsekakor pa je uporaba prototipov zelo priporočljiva in učinkovita pri izdelavi sistemov, kjer uporabniki še ne zmorejo natančno definirati, kaj naj bi sistem omogočal ali tam, kjer je razvoj zelo dinamičen in se zahteve uporabnikov nenehno spreminjajo. S pomočjo prototipov lahko enostavno in dosledno razvijamo vso potrebno dokumentacijo.

5.4 Metode uporabljene pri prototipiranju

Obstaja nekaj formalnih metod prototipiranja, čeprav je večina Agilnih metod močno odvisna od prototipnih tehnik.

5.4.1 Metoda dinamičnega razvoja sistemov – *DSDM*

Razvojna metoda dinamičnih sistemov je ogrodje za zagotavljanje poslovnih rešitev, ki se močno opira na prototipiranje kot temeljna tehnika in je sprejeta z *ISO9001*. Po *DSDM* je prototip lahko diagram, poslovni proces ali celo končni sistem v proizvodnji. Prototipi *DSDM* naj bi bili delni, od preprostih oblik pa vse do celovitih sistemov.

Prototipi *DSDM* so lahko grajeni tako po metodi zavračanja kot po razvojni metodi. Razvojni prototipi so grajeni tako horizontalno (širina nato globina) ali vertikalno (vsak odsek je zgrajen v detajle v naslednjih iteracijah pa so zgrajeni še preostali deli). Evolucijski prototip se sčasoma razvije v končni sistem.

DSDM priporoča naslednje vrste prototipov:

- ***Business* prototip** – uporabljajo se za načrtovanje in demonstracijo poslovnih procesov, ki se bodo avtomatizirali.
- **Uporabnostni prototipi** – uporabljajo se za opredelitev, izboljšavo, ter demonstracijo uporabnosti in izgleda grafičnih vmesnikov.
- **Zmogljivostni oziroma performančni prototipi** – z njimi želimo predvideti, kako se bo sistem odzival pod obremenitvenimi konicami. Prav tako se z njimi preverja ostale nefunkcionalne vidike sistema (stopnje transakcij, obseg shranjenih podatkov, odzivni časi, itd)

DSDM sestavlja naslednji življenjski cikel prototipa:

- Identifikacija prototipa
- Definicija plana
- Izgradnja prototipa
- Pregled in revizija prototipa

5.4.2 Operativno prototipiranje (*operational prototyping*)

Operativno prototipiranje je predlagal *Alan Davis* kot način integracije razvojne in metode zavračanja s konvencionalnim razvojem sistemov. Tako naj bi le dobro poznane značilnosti razvili po metodi razvojne metode, medtem ko bi ostale manj poznane funkcionalnosti razvili po metodi zavračanja prototipov.

Alanova ideja sloni na razvojni metodi, pri čemer prototip hitro pridobiva lastnosti končnega sistema, po vsaki fazi razvoja.

Metodologija sledi naslednjim korakom [13]:

- Razvojni model je grajen z uporabo običajnih razvojnih strategij, pri čemer se implementira le potrebne zahteve, katere so dobro poznane.
- Kopije prototipa in izkušene »prototiperje« pošljemo k večjim končnim uporabnikom, kjer jih nato ti spremljajo pri uporabi prototipa in beležijo eventualne težave.
- Ko uporabnik naleti na problem oziroma manjkajočo funkcionalnost, si to »prototiper« zabeleži. To razbremeni uporabnike zapisovanja težav. Zato uporabniki nemoteno nadaljujejo s pregledom prototipa.
- »Prototiper« nato posodobi popravke na prototipu, ter preda v ponoven pregled uporabniku. Če so popravki neefektivni, se jih takoj zavrže, če pa so bili popravki pravilni, »prototiper« dopolni tehnične zahteve, katere se posredujejo dalje razvojni ekipi.
- Razvojna ekipa prejme nove tehnične zahteve od vseh predanih tesnih kopij, ter tako z uporabo konvencionalnih metod izdela naslednjo stopnjo prototipa.

Jasno je, da je ključ te metodologije dobro usposobljeni prototiperji, ki so na voljo. Operativno prototipiranje ima veliko prednosti v zapletenih sistemih, ki imajo malo v naprej znanih zahtev.

5.4.3 Hitri evolucijski razvoj (Evolutionary rapid development ERD)

Koncept hitrega evolucijskega razvoja oziroma *ERD* temelji na ponovni uporabi delov sistema, uporabi programskih predlog in arhitekturnih predlogah. Proces se osredotoča na uporabo majhnih integracijskih ekip, ki delujejo paralelno s pogostimi interakcijami z naročnikom.

Ključ do uspeha, metodologije *ERD*, je v vzporedni raziskovalni analizi in razvoju funkcionalnosti, infrastruktur in komponent, kar omogoča hiter odziv na spremembe tehnologij, trga ali zahtev naročnika [9].

Okvir za sistem temelji na uporabi obstoječih- že objavljenih ali *de facto* standardih. Sistem je organiziran tako, da se omogoči razvoj sklopov zmogljivosti, ki vključujejo ugotovitve za izvedbo, zmogljivosti in funkcionalnosti. Omogočena je uporaba komponent, saj se ugotovljeni temeljni nabor funkcionalnosti, zelo verjetno, ne bo spreminjal. Bistvo cilja hitrega razvoja je uporaba metode »*timebox*«.

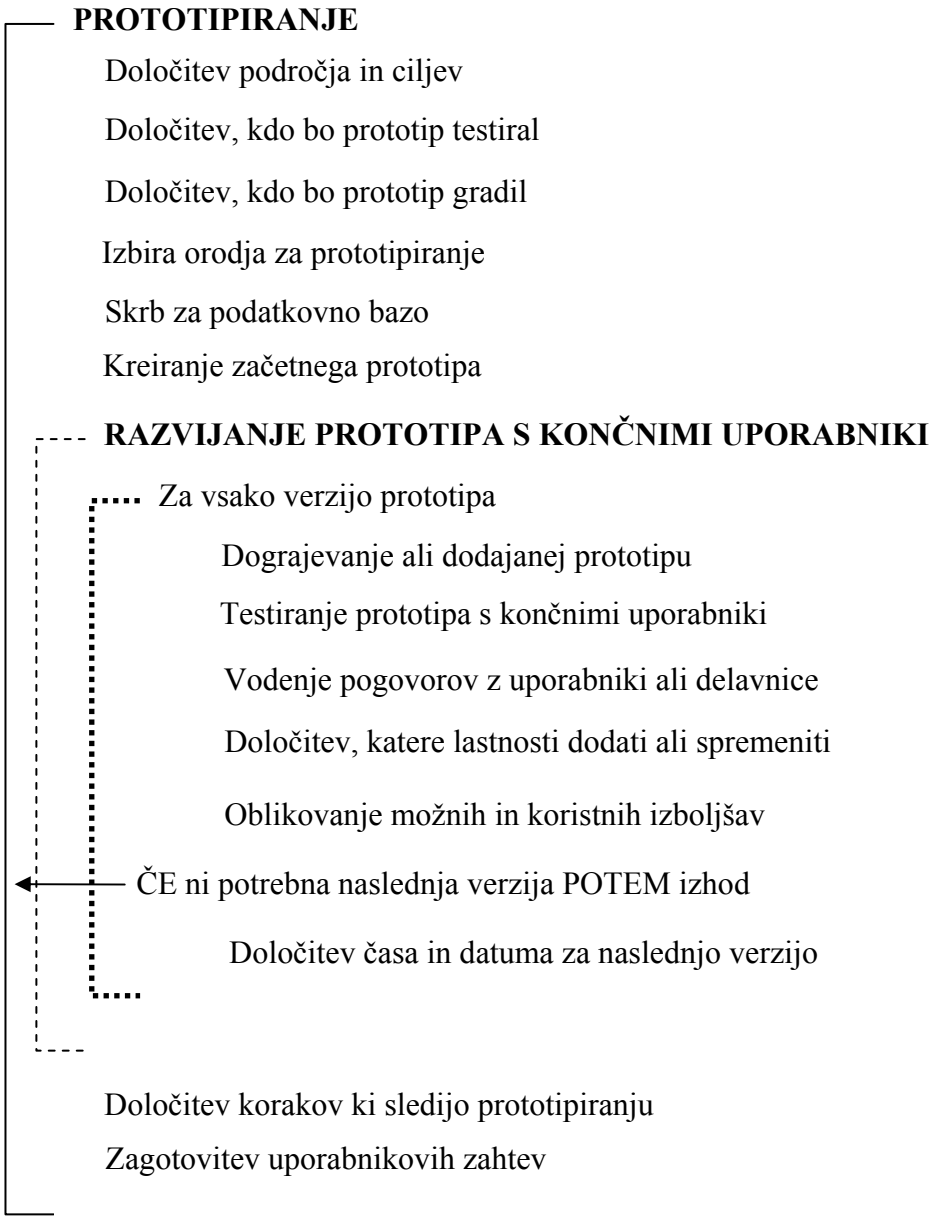
Metoda »*timebox*« temelji na časovnih obdobjih, v katerih je potrebno določene naloge (npr. razviti nabor funkcionalnosti) opraviti. Tako namesto, da se podaljšujejo roki zaradi izpolnjevanja nekaterih nejasno določenih ciljev, imamo pri tej metodi točno določene časovne mejnike (tako v smislu delovnih dni kot človek - ura). Cilji so zato lažje dosegljivi.

Ko imamo arhitekturo postavljeno, se programska oprema, njena integracija in preizkušanje, odvijajo na dnevni osnovi. Ekipa tako lažje ocenjuje napredek, ter hitro in objektivno prepozna morebitne težave.

Zaradi majhnih integracij sistema v določenem trenutku, je diagnosticiranje in odpravljanje napak, zelo hitra. Demonstracija naročnikom pa lahko poteka praktično ves čas razvoja, saj je prototip v vseh fazah izgradnje delujoč.

5.5 Postopek prototipiranja

Osnovna procedura prototipiranja naj bi vsebovala:



Slika 7: Osnovna procedura prototipiranja

Področje in cilji

Če je bila opravljena analiza s pomočjo informacijskega inženirstva, se lahko področje prototipiranja določi iz procesnih blokov v enciklopediji. Enciklopedija je nekakšna centralna shramba, v katero so vnesene vse sistemske informacije. V njej so shranjeni podatkovni modeli, procesiranja, informacije načrtovanja, dejstva, pravila ter nadzor organizacije in njenih sistemov.

Cilj sistema, ki ga prototipiramo, moramo določiti preden začnemo z načrtovanjem prototipa. Raziskati moramo ustrezne naloge, probleme in kritične faktorje, ki vplivajo na rezultat prototipa.

Prav tako je potrebno določiti, kako bo razdeljen in lociran bodoči sistem, slednje nam pomaga pri odločitvi, kdo bo prototip testiral.

Kdo bo prototip testiral

Poznamo dva načina testiranja:

- **validacija** – proces testiranja rezultatov v vsaki fazi življenjskega cikla. Pri tem se preverja ali se rezultati ujemajo z rezultati ugotovljenimi v predhodni fazi življenjskega cikla ter
- **verifikacija** – proces primerjave rezultatov sistema z uporabniškimi zahtevami.

Del procesa planiranja je tudi odločitev, kdo bo testiral pravilnosti delovanja prototipa. Glavni testerji so končni uporabniki, ki bodo končni sistem tudi uporabljali. Nujno je, da so nekateri testerji strokovno podkovani glede delovanja aplikacije.

Problem pri prototipiranju lahko nastane, če testerji niso dosledni ali nimajo motiva za natančen pregled prototipa. Zato moramo poiskati takšne, ki so dosledni in kritični.

Potencialni kandidati, ki sodelujejo pri testiranju so vodstvo, investitorji sistema, tehnično osebje, ki bo sistem gradilo, v nekaterih primerih tudi zunanji ljudje (kupci, zastopniki, ki so vpleteni v razvoj sistema), včasih pa vprašamo za mnenje tudi zunanje eksperte.

Kdo bo zgradil prototip

Običajno gradi prototip ena sama oseba, ki je dobro seznanjena z jeziki četrte generacije ali uporabljenim generatorjem. Ta oseba je običajno strokovnjak s področja informacijskega inženirstva, v nekaterih izjemnih primerih pa lahko kar končni uporabnik. Običajno strokovnjak izgradi prototip, končni uporabniki pa ga testirajo. Redki so primeri, vendar pa obstajajo, ko uporabniki zgradijo prototipe, testirajo pa ga strokovnjaki.

Prototip lahko gradi tudi več oseb, vsaka od njih z drugačnim znanjem. Ponavadi je to skupina strokovnjakov informacijskega inženirstva in uporabnikov. V splošnem ni priporočljiva gradnja prototipov z veliko skupino.

Za gradnjo prototipov naj bi bila zgornja meja dve osebi, saj v večini primerov to zadošča.

Gradnja prototipa

Rekli smo že, da mora biti v primeru uporabe razvojne metode prototipiranja skrbno načrtovana. Prototip ne sme postati izgovor za malomarnost pri delu in opustitev strukturnega načrtovanja. Če nismo dosledni, se lahko kaj kmalu zgodi, da dobimo namesto prototipa kaos, ki ga je težko spremeniti ali prevesti v sistem. Dobro prototipno orodje mora voditi k čistemu strukturnemu načrtovanju.

Pri prototipiranju moramo na začetku zgraditi preprost model, tako da že v razvojni fazi opazimo razhajanja med ciljem in nalogami. Pri zapletenih sistemih je potrebno najprej zgraditi njegovo ogrodje in šele nato vse ostalo.

Načini razvoja prototipov

Prototipe lahko razvijamo na dva osnovna načina:

- razvoj korak za korakom in
- nepretrgan razvoj

Pri razvoju korak za korakom sestavimo za vsak prototip listo potrebnih popravkov ali dodatkov. To ponavljamo tako dolgo, dokler ne dosežemo ciljne verzije sistema. Pri nepretrganem razvoju sodelujejo testerji, ki z graditelji prototipa sprotno le tega popravljajo. Tak način uporabimo takrat, ko obstaja močna povezanost med uporabniki in graditelji prototipa. Včasih je potrebno testiranje za nekaj časa prekiniti, da lahko graditelji preučijo staro verzijo in jo ponovno modificirajo.

Razlike med prototipom in sistemom

Ko končamo s prototipiranjem in izdelamo končno verzijo prototipa, se ta še vedno razlikuje od sistema. Po končanem prototipiranju sestavimo spisek lastnosti, ki jih prototip ne vsebuje, sistem pa jih mora.

Najpogosteje so:

- sposobnost okrevanja po napakah,
- varovanje podatkov,

- sposobnost revizije,
- enostavnost vzdrževanja,
- učinkovitost,
- ustrezni odzivni časi,
- dokumentacija.

Uporabnikom moramo razložiti, zakaj se prototip, takšen kot je, ne more uporabiti kot sistem. Prav tako jim moramo podati datumske okvire za dograjevanje sistema. Zgodi se namreč, da uporabniki ne razumejo, kako to, da je prototip tako hitro nastal, za izgradnjo sistema pa je potrebno dalj časa.

5.6 Prednosti prototipiranja

Obstaja veliko prednosti za uporabo prototipov pri razvoju programske opreme, nekatere so stvarne, ostale pa zgolj abstraktne.

Zmanjšanje časa in stroškov razvoja

Prototipiranje lahko izboljša kakovost zahtev in specifikacij razvijalcem. Ker se strošek sprememb v kasnejših fazah razvoja povečuje eksponentno, je zgodnje odkrivanje »kaj dejansko naročnik želi« izraženo v hitrejšem razvoju in cenovno ugodnejšem produktu.

Boljša in povečana udeležba uporabnikov

Prototipiranje zahteva udeležbo uporabnikov in jim dovoljuje vpogled in interakcijo s prototipom, kar jim omogoča boljše in kompletnejše povratne informacije in specifikacije [7]. Vpogled v prototip zmanjšuje možnosti nerazumevanja in slabega prenosa informacij, ki nastanejo, ko vsaka stran meni, da je druga razumela, kaj sama pričakuje in želi od produkta. Ker uporabniki bolje poznajo problemsko domeno, kot analitiki ali programerji, povečana interakcija med njimi pomaga k boljši kvaliteti končnega produkta. Pri takem načinu razvoja, je večja verjetnost da končni produkt bolje ustreza pričakovanjem naročnika tako po izgledu, kot po performančnih lastnostih.

Ostale prednosti prototipnega razvoja

Ostale prednosti prototipnega razvoja:

- uporabniki vidijo, kaj za njih gradimo ter podajo kritične pripombe,
- manjše tveganje, da zgradimo neustrezen sistem,
- vzpodbujanje uporabnikov h kreativnem delu oz. konstruktivnemu sodelovanju,
- za uporabnika bolje kot specifikacije na papirju,

- hitreje prototip kot specifikacije,
- napake lahko odkrijemo pred cenovno zahtevnimi fazami,
- navdušenost in večja zainteresiranost tako uporabnikov kot razvijalcev,
- preizkusimo lahko več različic.

Uporabljen na pravi način nam, prototipni pristop, zagotavlja hitrejši razvoj, kot tradicionalni življenjski cikel razvoja programske opreme.

5.7 Težave pri prototipiranju

Nezadostna analiza

Zaradi osredotočanja zgolj na določene funkcionalnosti, razvijalci ne posvečajo veliko časa za ustrezno analizo celotnega projekta. To lahko povzroči spregledanje boljše rešitve, pripravo nekompletne specifikacije ali slabo prevedbo prototipa v grajenje končnega produkta, ki jih je nato težko vzdrževati. Moramo se zavedati tudi, da je uporaba prototipa izven obsega njegove funkcionalnosti, odsvetovana. Včasih se namreč zgodi, da so razvijalci preveč posvečeni prototipu in njegovi izdelavi, kot končnemu modelu.

Razlikovanje prototipa in končnega sistema

Uporabniki v večini primerov smatrajo, da je prototip, kateri je namenjen ovržbi, uporabljal končni sistem, kateri potrebuje še določene popravke za dokončanje. Ne zavedajo se, da so aktivnosti preverjanja napak in dodatna varnostna zaščita zadeve, katere prototipi ne vsebujejo. Pričakujejo veliko od prototipa, kar pa ni namera razvijalcev. Velikokrat se tudi oprejo na funkcionalnosti, ki so sestavljale prototip, v končnem sistemu pa so bile odstranjene. V takih primerih lahko pride do konfliktov, saj zahtevajo te funkcionalnosti tudi v končnem sistemu.

Navezanost razvijalcev na prototip

Tudi razvijalci se lahko navežejo na prototip, na katerem so porabili veliko časa in napora pri izgradnji. To vodi do težav pri pretvorbi omejenega prototipa v končni sistem, ko nima ustrezno osnovane arhitekture. V takih primerih je priporočena uporaba metode zavračanja prototipov, saj v tem primeru do »skušnjave« ne more priti.

Prekomerna poraba časa za razvoj prototipa

Ključna lastnost prototipiranja je pričakovanje, da bo izvedeno hitro. Če razvijalci pozabijo na to dejstvo, bodo najverjetneje poskušali razviti prezapleten prototip. Naj na tem mestu dodamo tudi to, da ni vedno težava s prekomerno porabo na strani razvijalcev. Zna se zgoditi tudi, da se uporabniki zapletejo v debate o detajlih prototipa, kar privede preložitve končnega izdelka.

Stroški implementacije prototipa

Pogosta težava, pri sprejemanju prototipiranja, je visoko pričakovanje nad produktivnostjo, pozablja pa se na krivuljo učenja. Poleg usposabljanja za uporabo prototipne tehnike, se pogosto prezira na razvoj podjetij in potrebne podpore tehnologiji. Če pride do tega, se to odraža na nizki produktivnosti, kar odraža na višjih stroških proizvodnje [11].

6 Razvoj prototipa aplikacije

6.1 Ozadje

V podjetju *NGN Marko Kotnik*, *s.p.,* se ukvarjamo z razvojem spletnih aplikacij. Ker je na tem področju konkurenca neizprosna in veliko ponudnikov, smo odločili, da bodo naši produkti specializirani uporabnikovim zahtevam. Namreč na spletu obstajajo številne »odprto kodne rešitve«, katere ponujajo postavitev spletne strani praktično čez noč. Take rešitve so široko zasnovane, saj ponujajo vse v enem. V ponudbi takih rešitev, nismo videli tržne priložnosti, saj v njih ne vidimo dodane vrednosti, ter dolgoročnega obstoja na tržišču. V ta namen smo se lotili z razvojem lastnega sistem *CMS*. Ta temelji na načelu »ponuditi naročniku točno to, kar potrebuje« in ne kopice dodatnih nastavitev in dodatkov, za katerih končni uporabnik ne potrebuje.

Tukaj smo naleteli na »oviro«. Ponujene rešitve omogočajo postavitev spletne strani v par korakih, kar pa specializirana rešitev, kar naš produkt tudi je, nikakor ne. Stranke so v večini primerov laiki. Veliko se jih želi posluževati spletnih strani, ker so slišali, da je to dobro imeti. Dolgotrajni razvoj sistema odpade, saj naročnik potrebuje hitro rešitev in ne take, ki bi bila operativna šele čez nekaj mesecev.

Pri razvoju spletnih aplikacij je bilo zato potrebno analiziranje in načrtovanje novega sistema časovno zmanjšati na minimum, ter stranki v čimbolj zgodnji fazi razvoja ponuditi rešitev. S tako rešitvijo pa jo nato pripraviti do vzajemnega sodelovanja vse do končnega produkta in postavitve sistema na svetovni splet.

V podjetju smo se odločali med različnimi metodologijami razvoja, odločitev je pri tem padla na metodologijo prototipiranja. Ta se je v večini primerov nato izkazala za zelo uporabno. Razvojna ekipa je skozi celotni življenjski cikel v kontaktu z naročnikom, kateri lahko spremlja projekt skozi vse faze razvoja od oblikovanja grafičnega vmesnika na papirju, pa do končne rešitve.

Tak pristop se je izkazal za učinkovitega tudi, zaradi hitre postavitve začetnih prototipov spletne strani v njenem začetku. Izkušnje iz preteklih projektov so pokazale, da je vnos in urejanje vsebine nove spletne strani v začetku, precej nepriljubljeno opravilo. Prototipiranje pa naročnika k temu dejansko prisili, saj takoj, ko je prototip administracije postavljen, že zahteva od naročnika vnos podatkov ter preveritev delovanja pred nadaljevanjem izdelave naslednjih korakov. Prav tako prototipiranje skrajša razvoj, saj velikokrat se zgodi, da so zahteve naročnika nejasne. V takih primerih gre velikokrat razvoj v napačno smer, kar pa z izbrano metodologijo hitro ugotovimo, ter tako stroške povezane z napačno izbiro hitro omejimo.

Na tem mestu velja tudi pripomniti, da prototipiranje zahteva od razvojne ekipe veliko dodatnega napora, predvsem v primerih, ko ima naročnik nejasne cilje. V takih primerih mora ekipa s svojimi izkušnjami svetovati izgled in funkcionalnosti bodoče aplikacije, ter s tem naročnika prepričat v smotrnost projekta.

V nadaljevanju bom metodologijo prototipiranja prikazal na primeru razvoja spletnega portala. Predstavil bom uporabljena orodja, ter uporabljene postopke ter knjižnice. Zaradi obsežnosti projekta bom za ta namen predstavil zgolj del končnega prototipa.

6.2 Opis aplikacije

Spletni portal bo namenjen naprednemu iskanju pri nakupu novega avtomobila. Konkurenčni portali ponujajo iskanje zgolj po parametrih, katere pa mora vsak obiskovalec vnaprej poznati (npr. proizvajalec, model, cena od, cena do, itd.). Naša rešitev bo naročniku omogočala vnos avtomobilov, obiskovalcem portala pa na enostaven način preko izbire tipa avtomobila (npr. športni, limuzina, kabriolet, itd.), ter cenovnega razpona, ki jim ga bo ponudi portal pregled nad izbranimi modeli. Obiskovalec bo lahko dodatno podal še subjektivno pomembnost dodatnih parametrov (npr. stroški vzdrževanje, onesnaževanje, pospeški, itd.) ter tako prišel do ožjega nabora modelov, kateri bodo najbolj ustrezali njegovim kriterijem.

Taka rešitev bo na področju iskanja avtomobilov inovativna. V večini primerov, namreč, večina uporabnikov ne pozna novih modelov in znamk, pozna pa kriterije, katerim naj bi jih bodoči avto moral zadovoljevati. S tako rešitvijo bi zato naročnik pridobil konkurenčno prednost pred konkurenti, predvsem v segmentu nepoznavalcev znamk in avtomobilov.

6.3 Problemska domena

Naročniku je potrebno ponuditi rešitev naprednega iskalnika po parametrih. Za vnos parametrov je potrebno zagotoviti podatkovno bazo. Ker se v vnaprejšnjega števila parametrov ne pozna, je nujnost izgradnje dinamičnega administracijskega vmesnika znotraj obstoječe *CMS* rešitve.

Ker so določeni parametri grupirani na nivoju znamke avtomobila, ostali pa so specifični za določeni model je potrebno zasnovati rešitev, ki bo podpirala naročnikove zahteve.

Zaradi naprednosti zahtev, je potrebno razviti osnovni prototip, v naslednjih korakih pa ga nadgrajevat novim zahtevam, saj tudi naročnik nima jasno definiranih prioritet. Tako bo razvojna ekipa neprestano v kontaktu z naročnikom, ter naročnikove zahteve hitro v celoti upoštevala.

Rešitev znotraj administracije mora upoštevati kasnejše nadgradnje, prav tako morajo biti vnosi v podatkovni bazi optimizirani za iskanje, saj se pričakuje nekje okrog 10000 vnosov, za vsak vnos pa reda 40 parametrov.

6.4 Uporabljena orodja in knjižnice

Ker ima razvojna ekipa že veliko izkušenj na tem področju, na tem mestu ni bilo potrebe po izbiri novih orodij. Za programski jezik, na katerem bo tekla spletna aplikacija smo se odločili za jezik *PHP*, za podatkovno bazo, pa je odločitev glede na naročnikove zahteve padla na podatkovni strežnik *MySQL*. Odločilno je k obema odločitvama botrovalo to, da sta obe »odprto kodni«, zaradi česar ne bo v kasnejših fazah prihajalo do dodatnih stroški povezani z nakupi licenc.

Sodobne spletne aplikacije stremijo k uporabi tehnologij tipa *web2.0*, kar pomeni visoko interakcijo z uporabnikom na spletu, to hkrati pripelje do uporabe *Javascript* in z njim povezane tehnologije *AJAX*.

V preteklih projektih se je na tem mestu izkazala knjižnica *jQuery* za odlično rešitev. Saj ponuja vse prednosti *Javascript*-a, kot *AJAX*-a. Njen slogan »*write less, do more*«, pa nas je z njenim nenehnim razvojem in vnaprej napisanimi rešitvami ter enostavno uporabo, prepričal.

Pri prototipih spletnih aplikacij gre za pretvorbo iz grafične oblike v delujočo aplikacijo. Ponavadi smo v prvih korakih osredotočeni na razrez grafičnega vmesnika v *HTML* predstavitev brez dodanih funkcionalnosti. Nato se strani *HTML* dodatno izpopolnjuje, vse do prevedbe v *Smarty PHP*. Zanj smo se odločili, saj nam je pomembno, da je predstavitveni nivo ločen od aplikacijskega. Tako hitreje in enostavneje najdemo napake v kodi, poleg tega pa je kasnejše vzdrževanje bistveno hitrejše.

Ker smo se v predhodnih fazah odločili za *PHP* je bil *Smarty* idealna rešitev. Deluje namreč »platformi *PHP«,* istočasno pa je vanj vključitev »*HTML* kode«, generirane iz predhodnih faz, precej hitra in enostavna.

Končna spletna rešitev bo tekla na spletnem strežniku *Apache HTTP server*.

Podrobnosti o ključnih odločitvah za posamezno orodje in njegovo delovanje v nadaljevanju.

Apache HTTP server

Spletni strežnik *Apache* [24] igra ključno vlogo pri razvoju spleta. Bil je prva alternativa *Netscapeovem* spletnem strežniku, trenutno znanemu kot spletni strežnik *Sun Java System*. Od aprila 1996 je *Apache* najbolj uporabljen strežnik *HTTP* na celotnem spletu. Leta 2007 je bilo na *Apache* strežnikih postavljenih približno 48 % vseh spletnih strani.

Aplikacija je bila sprva zgrajena za operacijski sistem *Unix* in *FreeBS*D, danes pa je na voljo v večini operacijskih sistemov, kot so *Microsoft Windows, Solaris, MAX OS-X* idr.

Relacijska podatkovna baza: MySQL

MySQL [23] je sistem namenjen upravljanju z relacijskimi podatkovnimi bazami. Gre za eno od odprtokodnih rešitev implementacije relacijske podatkovne baze, ki za delo uporablja jezik *SQL*.

Deluje na principu odjemalec – strežnik, pri čemer je strežnik lahko nameščen kot sistem porazdeljen na večjem številu strežnikov.

Glavne značilnosti podatkovnega strežnika *MySQL* so enostaven nabor ukazov, zmogljivost (dovolj dobra za večino spletnih aplikacij) in brezplačnost.

PHP

PHP je odprto kodni strežniški skriptni jezik, ki se uporablja v povezavi s *HTML*-jem. Kot del *PHP*-ja sta podprti tudi možnosti zaganjanja skript v ukaznem načinu in kreiranje grafičnih aplikacij, za kar ga bomo tudi mi uporabljali v nadaljevanju. V začetku je bil zamišljen kot niz *makrojev*, ki bi razvijalcem pomagal pri vzdrževanju osebnih domačih strani. Začetki segajo v leto 1994, ko ga je napisal dansko-kanadski programer *Rasmus Lerdord*, da bi zamenjal nekaj v *Perlu* napisanih skript. Od takrat so bile možnosti *PHP*-ja razširjenje, iz niza pripomočkov v programski jezik, s številnimi možnostmi, s katerimi je mogoče upravljati tudi velika spletna okolja.

Smarty PHP

Smarty je »template« sistem, ki poskuša ločiti logiko programa od prezentacije. Glavni namen *Smarty*-ja je torej, da nam pomaga pri sami prezentaciji oziroma izpisu vsebine.

Uporaba *Smarty*-ja v večini primerov preprosta, predvsem pa pride do izraza v primerih, ko se razvoj aplikacij razdeli na delo razvijalca in oblikovalca.

jQuery

jQuery je ena izmed najbolj popularnih *Javascript knjižnic* na spletu, poganja pa zelo širok spekter različnih spletnih strani; od medijsko bogatih, interaktivnih in efektov polnih strani, do poslovno kritičnih aplikacij z naprednimi vmesniki.

Ponuja nam interakcijo med *JavaScript*-om in *HTML*-jem. Izdan je bil januarja 2006, trenutno pa ga uporablja že 27 % od 10.000 najbolj obiskanih strani. Omenja se, da naj bi bila najbolj popularna *JavaScript knjižnica* dandanes [25].

jQuery je bil razvit, da bi olajšal navigacijo skozi *DOM* elemente *HTML*-ja, izdelavo animacij, upravljanje dogodkov in razvoj *AJAX* aplikacij. Razvijalcem je omogočil izdelavo naprednih učinkov na visoki ravni ter interakcijo z animacijo.

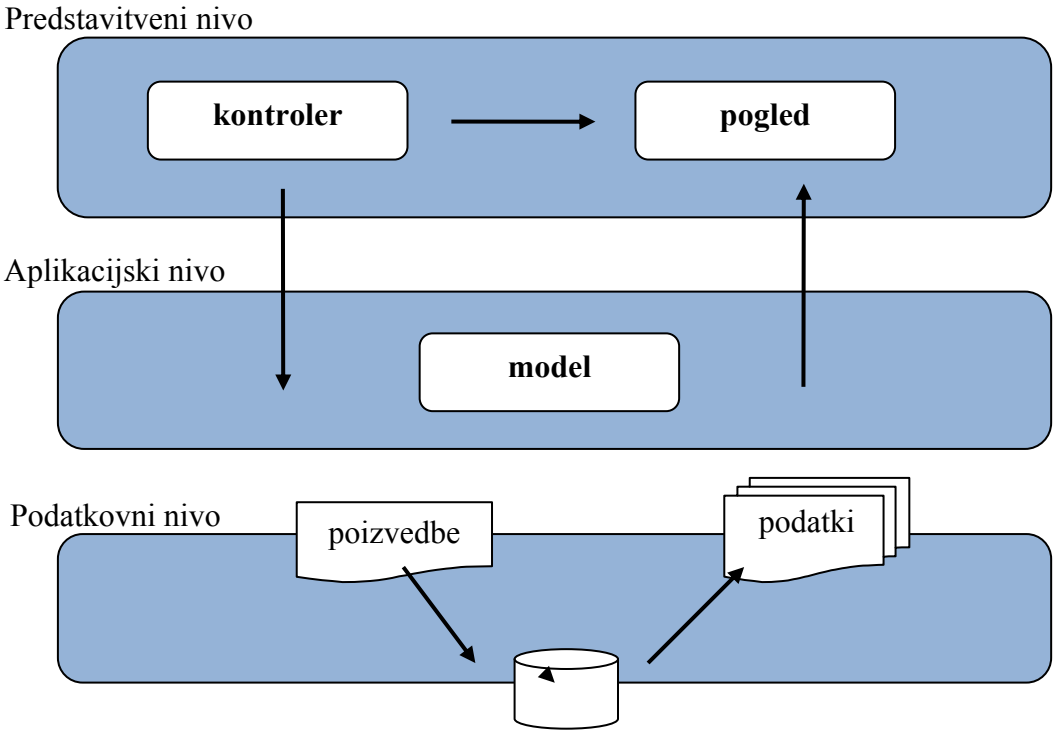
Kot vidimo, iz zgornjih opisov, so nekatere odločitve, kot na primer *zakaj izbrati določeno orodje?,* trivialne. Nekatere so sicer botrovale predhodnim izkušnjam, v vseh primerih pa stremimo k učinkoviti in hitri uporabi, kjer morajo biti rezultati vidni v kratkih intervalih ter merljivi.

6.5 Arhitektura

Prototip spletne aplikacije bo tekel na že predhodno zgrajenem *CMS* sistemu. Zato bo moral podpirati predhodno razvite protokole preverjanja in validacije. Želimo doseči tudi tako stopnjo integracije, da bo lahko v prihodnje uporabljen tudi na drugih projektih z malenkostnimi modifikacijami. Prav tako je že v prvi fazi zasnovan na podpori večjezičnosti, kar bo v nadaljevanju razvidno iz entitetnega diagrama.

MVC

MVC (*Model-View-Controller*) je arhitektura, ki deli aplikacijo na tri nivoje: model, pogled in kontroler.



Slika 8: Arhitektura MVC kot del trinivojske arhitekture [26]

Kontroler spremlja in se ustrezno odziva na dinamiko procesiranja. Pri spletnih aplikacijah se na podlagi spremenljivk *GET* ali *POST* odzove z ustreznimi akcijami.

Model vsebuje logiko za dostop do podatkov na s katerimi aplikacija razpolaga. V primeru zahteve po shranjevanju podatka, vsebuje logiko za preverjanje ustreznosti posameznih parametrov.

Pogled skrbi za končni prikaz uporabniku, ponavadi v obliki uporabniškega vmesnika, ni pa nujno. Pri spletnih aplikacijah je to v večini primerov *HTML*.

V našem primeru bomo prototip gradili po vzoru arhitekture *MVC* kot del trinivojske arhitekture.

7 Prototip v praksi

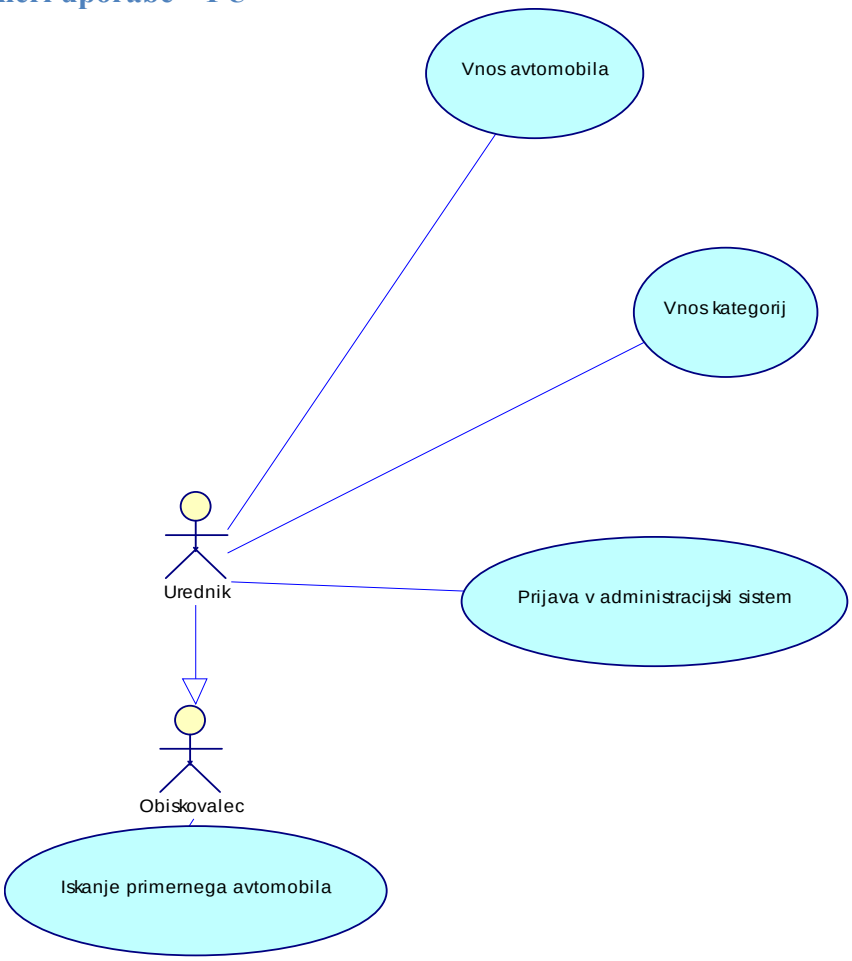
Najprej je bilo potrebno, v podatkovni bazi *MySQL,* kreirati ustrezne tabele. Nato sem generiral potrebne *XML* datoteke za pravilno delovanje novega prototipa na obstoječem sistemu. Glede na to, da je bil *CMS* sistem predhodno razvit, tako da le ta ustreza skalabilni uporabi, sem se predhodno odločil, da je najboljša rešitev ta, da so nastavitve posameznega modula shranjene v *XML* datoteki. Tukaj so predvsem imena tabel v podatkovni bazi, imena modula in način sortiranja stolpcev. Tako lahko iz modula za novice hitro naredimo modul za ankete. Pri tem obdržimo predhodne funkcionalnosti vnašanja v podatkovno bazo, urejanja ter brisanja in po potrebi dodamo nove funkcionalnosti.

Ko smo slednje uredili, smo pripravili še potrebne predloge *Smarty*. To so *HTML* datoteke iz katerih bomo v naslednji fazi kreirali vnosno formo.

Z uporabo programov četrte generacije sem za lažje razumevanje problemske domene generiral ustrezne diagrame.

7.1 Prototip 1.1

7.1.1 Primeri uporabe – PU



Slika 9: Diagram primerov uporabe prototipa 1

Diagram primerov uporabe jasno prikazuje ločnico med urednikom in obiskovalcem. Urednik ima zgolj pravice urejanja podatkov, medtem, ko obiskovalec lahko zgolj pregleduje predhodno vpisane parametre.

7.1.2 Primer uporabe vnos avtomobila

7.1.2.1 Kratek opis

Vnos avtomobila z vsemi potrebni parametri v sistem.

7.1.2.2 Glavni tok

1. Administrator izbere dodajanje avtomobila
2. Sistem prikaže vnosno formo s polji:
    a. Naziv avtomobila
    b. Kategorija avtomobila

   c. Ali bo avtomobil nemudoma viden na strani
   d. Datum od katerega dalje se avtomobil prikazuje
   e. Opis avtomobila
   f. Opcija dodajanje glavne slike avtomobila
   g. Opcija dodatnih slik
   h. Opcija *SEO* opisov
3. Urednik izpolni zahtevane podatke in potrdi vnos
4. Sistem preveri vnose ter shrani zapis v podatkovno bazo
5. Sistem preusmeri urednika na novo vnosno formo, kjer se vnesejo dodatni parametri.

7.1.2.3 Alternativni tok

7.1.2.3.1 Dodaj kategorijo

Želene kategorije ni v naboru kategorij.

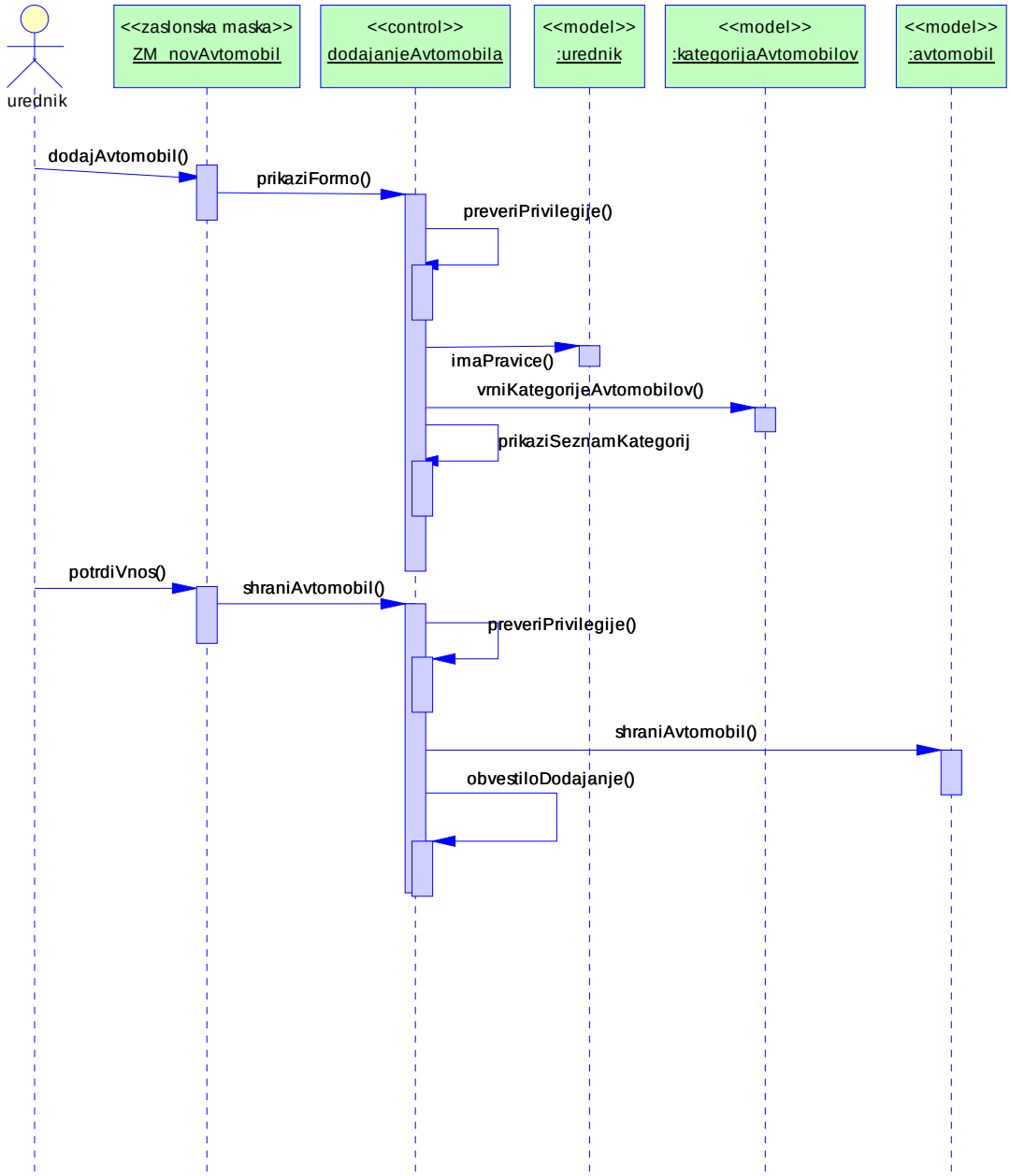
Administrator izbere zaslonsko masko dodajanje nove kategorije.

1. Sistem prikaže trenutno vnesene kategorije. Administrator izbere kategorijo, katere sina bo kreiral v naslednjem koraku.
2. Sistem pokaže vnosno formo za vnos kategorije s polji:
   a. Naziv kategorije
   b. Ali bo kategorija nemudoma vidna na strani
   c. Opcija dodajanja slike kategorije
   d. Opis slike
   e. Opcija *SEO* opisov
3. Administrator vnese zahtevane podatke in potrdi vnos
4. Sistem preveri vnose ter shrani zapis v podatkovno bazo
5. Sistem preusmeri urednika na novo vnosno formo, kjer se vnesejo dodatni parametri kategorije.

7.1.2.4 Posebne zahteve

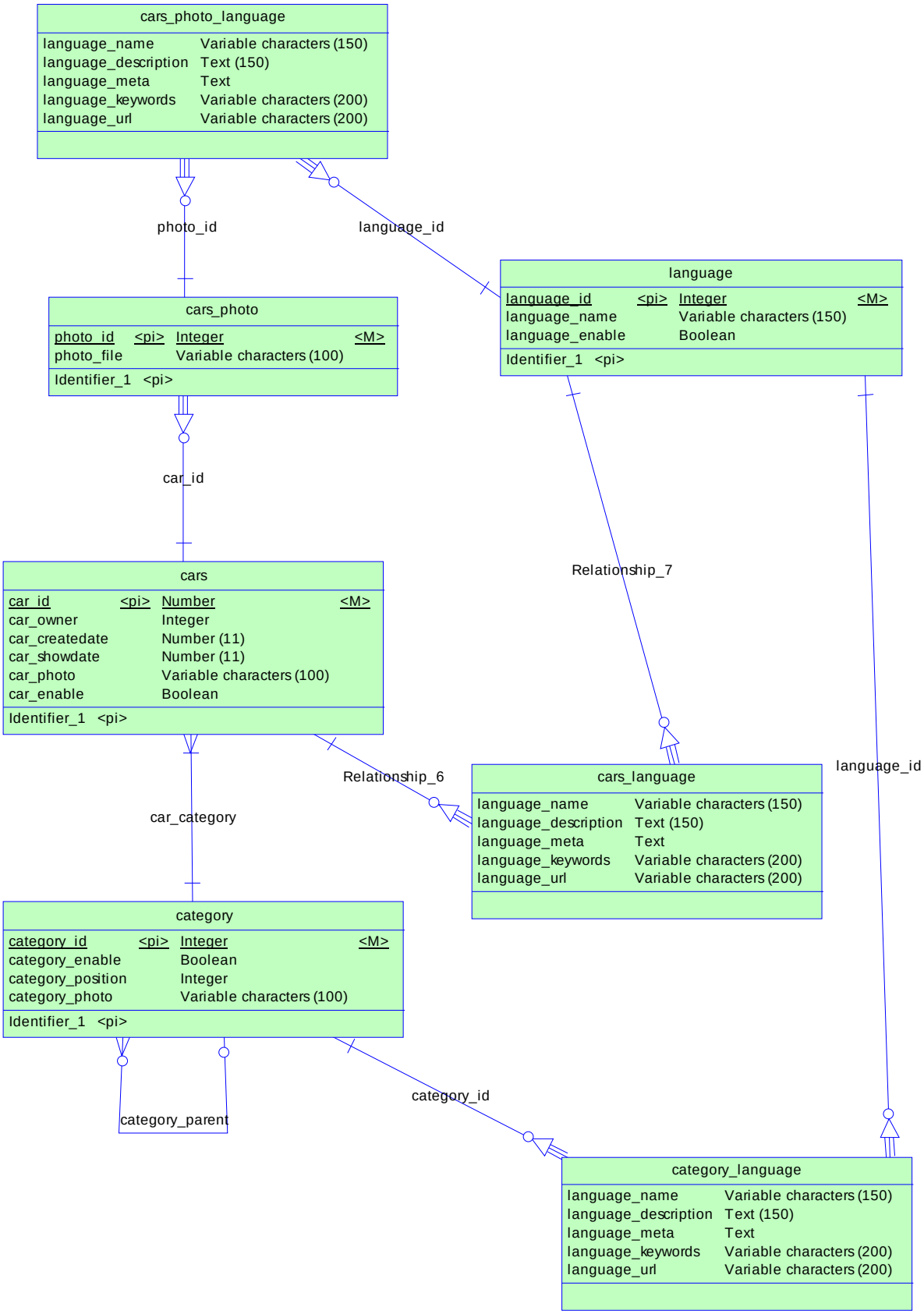
Urednik, ki dodaja tako kategorijo, kot avtomobil, mora imeti privilegije nad izbranim modulom. Sistem mora avtomatsko zmanjšati izbrane slike na ustrezne velikosti. Podprto mora biti dodajanje dodatnih parametrov, tako kategorijam kot avtomobilom.

7.1.3 Diagrami zaporedja



Slika 10: Diagram zaporedja za primer uporabe dodaj avtomobil

7.1.4 Entitetni diagram podatkovne baze

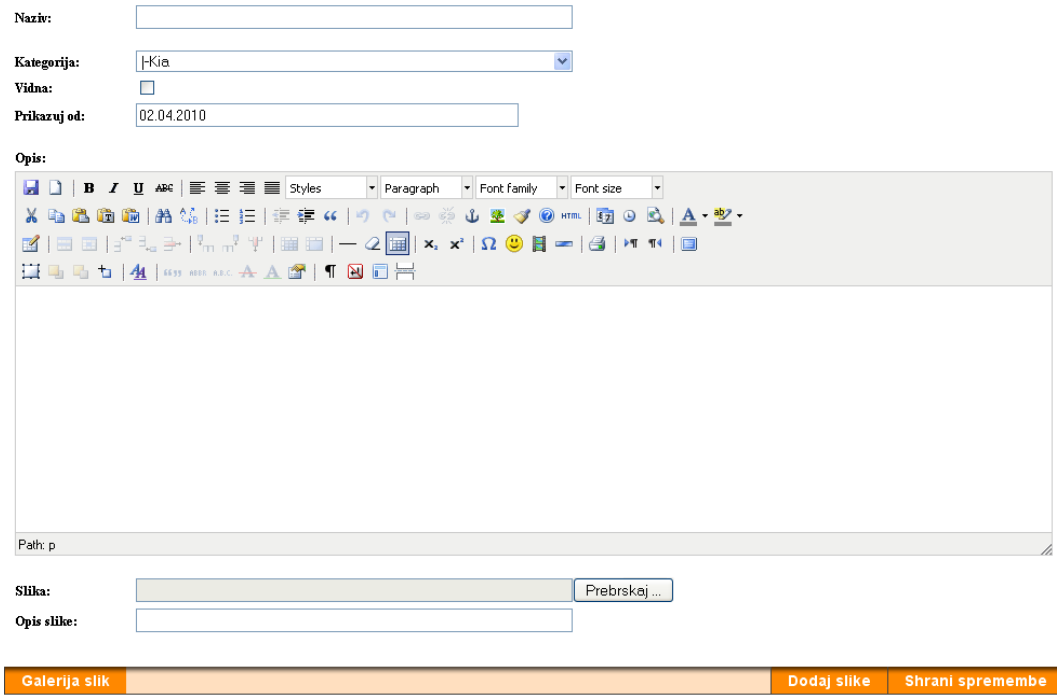


Slika 11: Entitetni diagram

7.1.5 Zaslonska maska vnos avtomobila

Zaslonska maska mora ustrezati zahtevam vnosa novega avtomobila. Predhodno sem opisal to z diagramom zaporedja ter glavnim in alternativnim tokom. Ker gre za prototip, stranka pa še ni dokončno definirala funkcionalnosti bodočega sistema, smo v tej fazi prepuščeni razvoju prototipa po najboljših praksah in izkušnjah iz preteklih projektov.

Nastala je naslednja zaslonska maska. Slednja ustreza glavnemu toku vnosa novega avtomobila v sistem.

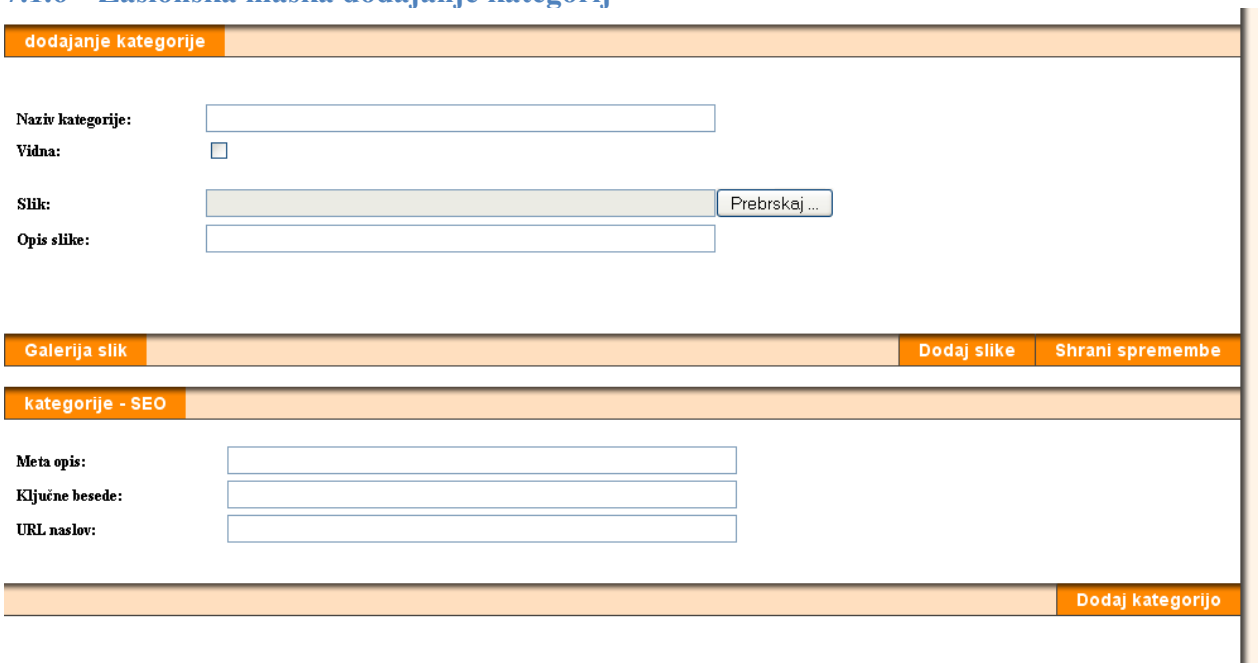


Slika 12: Zaslonska maska prvotnega prototipa

Vnosna forma ustreza vsem zahtevam vnosa novega avtomobila, vendar za nemoteno uporabo spletne aplikacije potrebujemo tudi vnosno formo za dodajanje kategorij.

Ker bo stranka s prvim prototipom dobila zgolj občutek delovanja bodočega sistema, v tem koraku ne smemo pretiravati s funkcionalnostjo, saj se kmalu lahko zgodi, da prototipiranje ne bi imelo svojega namena. Izgubili bi se v prekomerni porabi časa potrebnega za razvoj prototipa.

7.1.6 Zaslonska maska dodajanje kategorij



Slika 13: Zaslonska maska dodajanje kategorij

S sledečo zaslonsko masko smo želeli omogočiti uredniku nemoteno vnašanje podatkov v podatkovno bazo. Na ta način smo zadovoljili potrebo po vnosu kategorij.

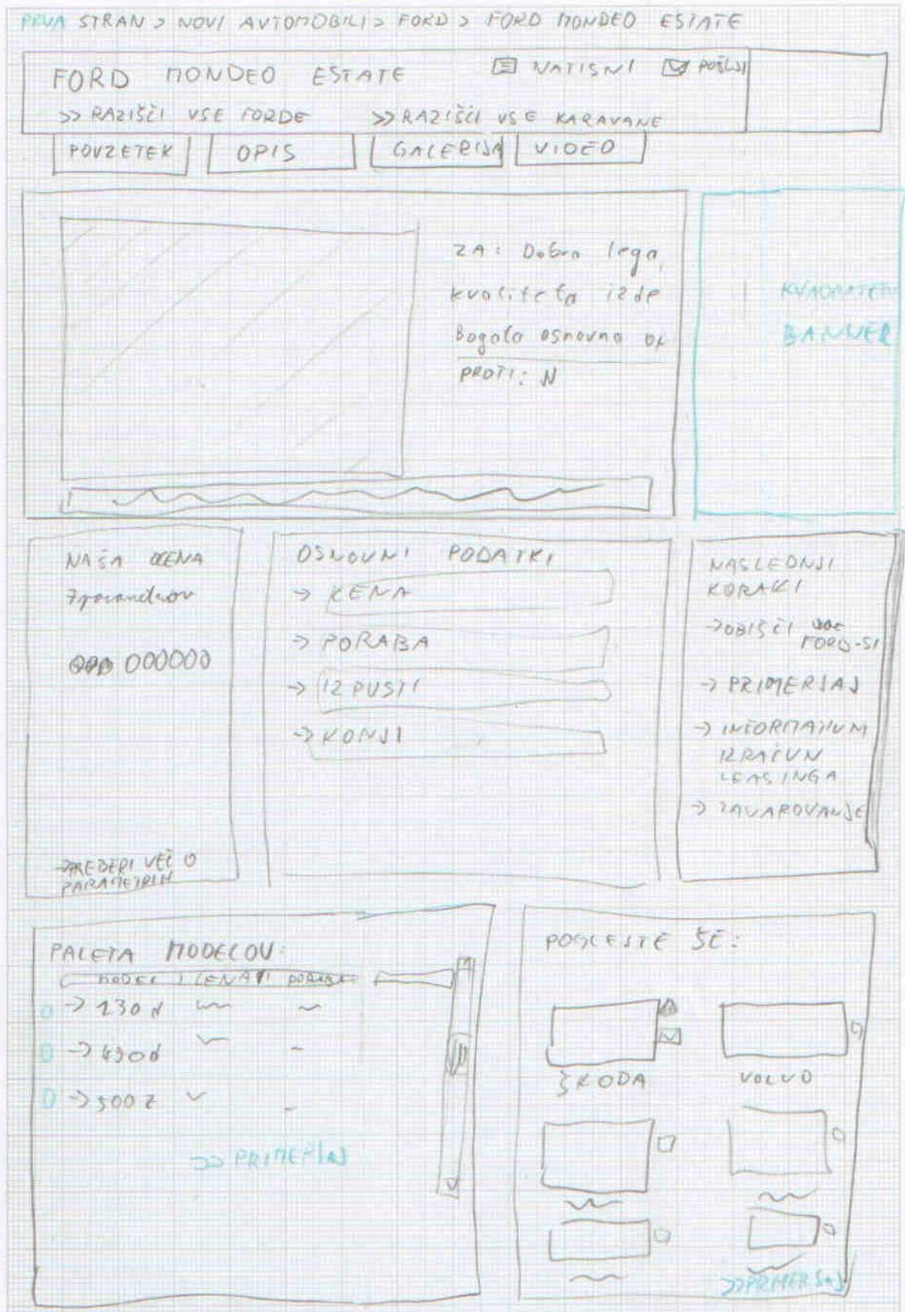
Sledi sestanek z naročnikom, predstavitev prototipa ter iskanje novih funkcionalnosti ter pomanjkljivosti različice *prototipa 1.1*.

7.2 Prototip 1.2

Na sestanku je naročnik podal parametre, katerim bo bodoči sistem moral zadovoljiti. Pregledali smo *prototip 1.1*, ter ugotovili naslednje spremembe, katere bo potrebno implementirati v novi verziji:

- Podpora kategorij v drevesni strukturi (npr. *Ford -> Ford Mondeo*)
- Ugotovljeni so bili osnovni parametri, ter njihovo število (ključni dejavnik uspeha)

Definiran je bil naslednji mejnik v izdelavi ter časovni roki. Na sestanku je nastal tudi dokument izgleda bodočega vmesnika za obiskovalce spletnega portala.

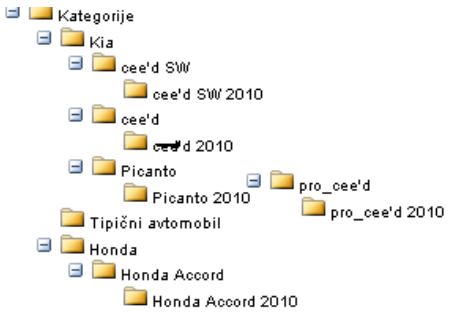


Slika 14: Skica grafičnega vmesnika za prikaz avtomobila obiskovalcem

7.2.1 Zaslonska maska dodajanje kategorij

Na predhodno zgrajeni zaslonski maski je bilo potrebno podpreti funkcionalnosti postavitve kategorij v drevesno strukturo (definirati očete in sinove).

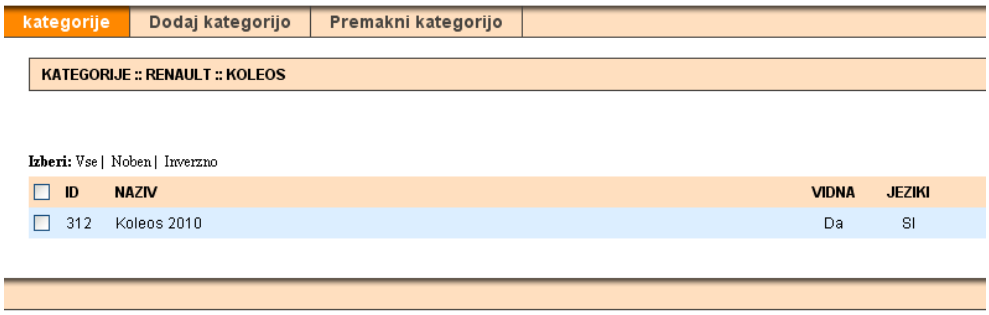
V ta namen smo na tem koraku razvili premikanje med vnosi v globino. Prav tako smo razvili možnosti spreminjanja vrstnega reda kategorij.



Slika 15: Možnost spreminjanja vrstnega reda kategorij

Na spodnji zaslonski maski je prikazana opcija dodajanja podkategorij. Dodali smo podkategorijo *Koleos 2010 očetu Koleos*.

Seznam podkategorij je prikazan v tabelarični obliki poleg pa so dodani še dodatni atributi.

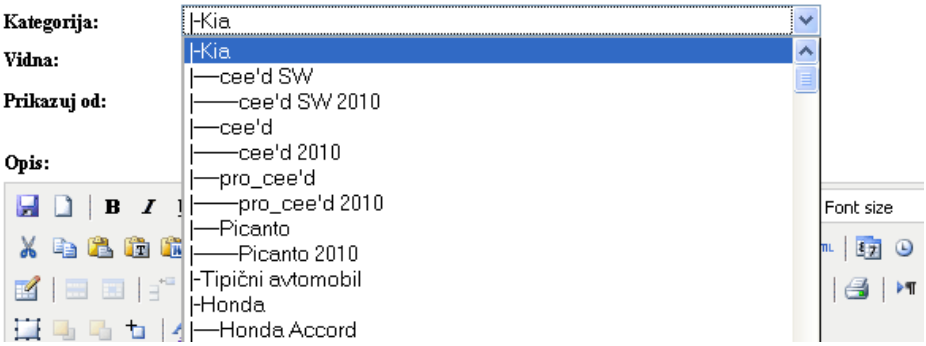


Slika 16: Možnost dodajanja sinov kategorijam

7.2.2 Zaslonska maska dodajanje avtomobila

Na tem mestu je bilo potrebno dodane funkcionalnosti, ki so bile dodane na modelu kategorije avtomobilov podpreti še na nivoju zaslonske maske dodajanje avtomobila.

Na nivoju izbire kategorije smo vnose predstavili na nivoju podkategorij. Tako lahko urednik nemoteno izbere vrhnjo kategorijo oziroma podkategorijo za posamezni avtomobil.



Slika 17: Zaslonska maska dodajanje avtomobila z možnostjo izbire med podkategorijami

7.2.3 Izdelava grafične predloge zaslonske maske iskalnika

Ker gre tudi pri izdelavi grafične predloge maske bodočega sistema za prototipni razvoj, bomo na tem mestu to tudi prikazali v praksi.

Vodja razvojne ekipe je po vseh zbranih zahtevah naročnika oblikoval splošno mnenje, kako naj bi bodoči spletni portal izgledal. Zahteve so bile posredovane grafičnemu oblikovalcu. Ta je bil z vodjo v konstantnem stiku, tako je vodja po vsaki večji spremembi dobil v predogled in diskusijo novo različico grafičnega vmesnika.

Slika 18: Izdelava grafične predloge na principu prototipiranja

Pri izdelavi grafične predloge je nastalo preko 70 različnih prototipov. Pri tem je predvsem šlo za pozicioniranje posameznih elementov na ustrezna mesta znotraj grafičnega vmesnika. Prav tako je bilo potrebno z grafično predlogo pokriti vse potrebne funkcionalne zahteve.

7.3 Prototip 1.3

V pretekli iteraciji prototipa so bile dodane dodatne funkcionalnosti na področju kategorizacije avtomobilov. Prav tako je bila izdelana grafična zasnova iskalnika. O obojem bomo v tej interakciji predebatirali z naročnikom ter določili nove zahteve za novo različico prototipa.

Naročnik je imel nalogo, da do sestanka pripravi podrobnejši seznam bodočih parametrov avtomobila, ter da jih omeji na katerem nivoju bodo zbrani. Nastal je dokument, na primeru avtomobila *Ford Mondeo 1.8 TDCi* [Tabela 1].

Zaradi narave diplomskega dela, smo nekatere odstranili ter predstavili zgolj osnovne za lažje razumevanje.

Parametri so grupirani na nivoju modela (roza obarvano), ter kasneje še dodatno na nivoju različice modela (modro obarvano).V ta namen je bilo potrebno razviti ustrezni prototip, da bo uspešno podprl novim zahtevam naročnika.

Dodatno je bilo potrebno upoštevati, da bodo nekateri parametri zbrani zgolj na nivoju različice, drugi pa le na nivoju modela. Dodatna zahteva je bila tudi ta, da bodo nekateri parametri na nivoju modela zgolj akumulativne vrednosti njegovih različic.

Po pregledu obstoječega entitetnega diagrama, smo ugotovili, da je tega potrebno dopolniti, saj nove zahteve niso bile podprte. Dodati je bilo potrebno nove entitete, ter omogočiti dodajanje parametrov preko administracijskega vmesnika. Tako bi lahko enostavno zagotovili kasnejše dodajanje novih parametrov, ne da bi bilo zaradi tega potrebno posegati v programsko kodo.

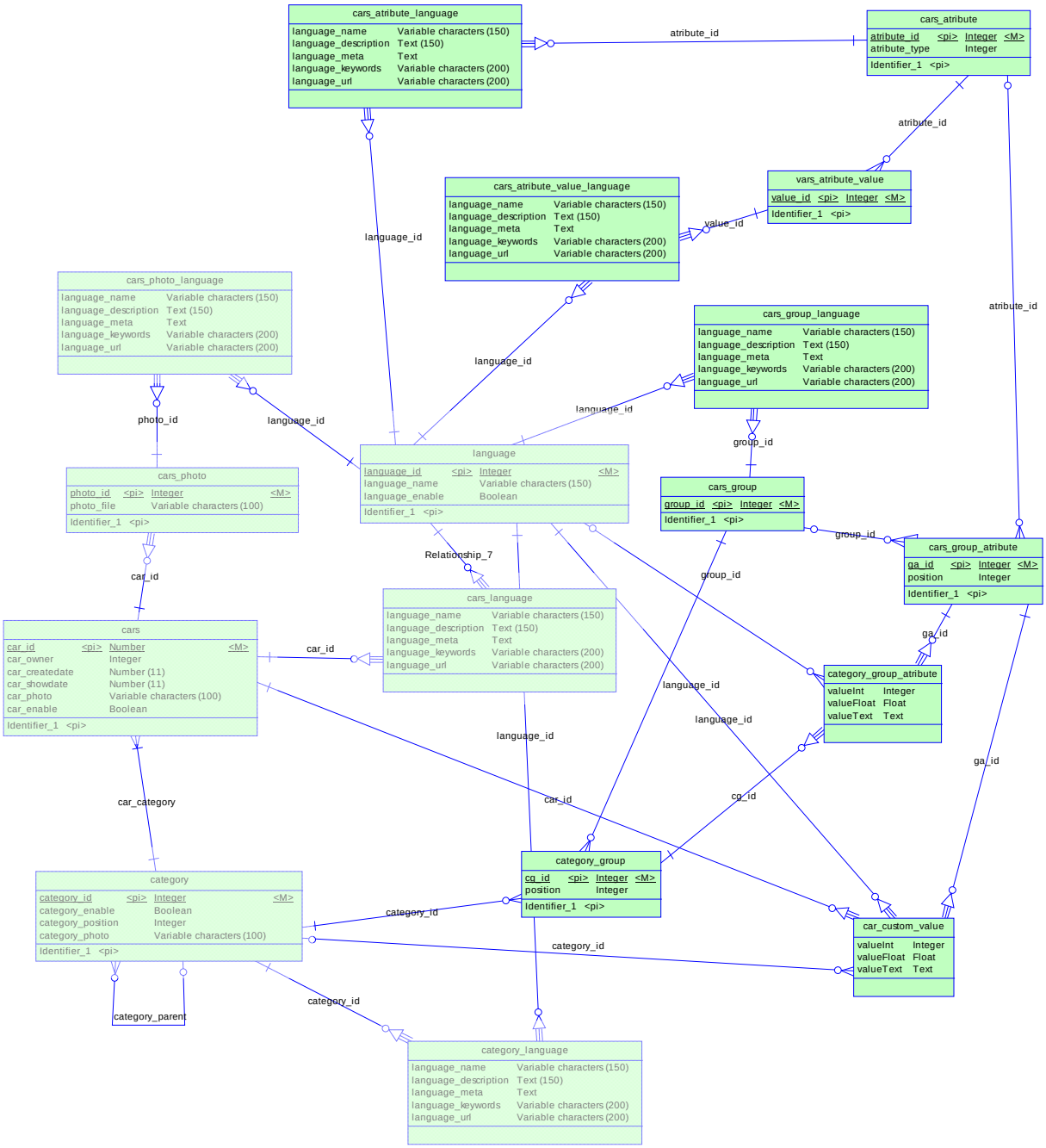
Na tem mestu smo se dogovorili o potrebnih zahtevah do naslednjega sestanka, ter opravilih katere je potrebno storiti, tako na strani razvojne ekipe kot na strani naročnika.

7.3.1 Tabela zahtev

	Ime vnosa	Ford Mondeo 1.8 TDCi	
	Znamka	Ford	txt
	Opis znamke	3 povedi	txt
	Zastopniki	seznam zastopnikov, razporejenih po regijah	txx
	Novice in videi	povezujemo jih na znamko	
	Model	Mondeo	txt
	osnovni podatki so črpani iz motorne različice		
	kategorija	enako kot motorna različica	
	poraba	povprečje	
	cena	razpon	
	moč	razpon	
	Parametri za čarovnika		
	Udobje	7	Prva cifra je značilnost modela(1-10),
	izpusti		
	vodljivost		
	prostornost		
	hitrost in pospeški		
	kvaliteta izdelave		
	stroški		
	Opis avtomobila		
	Za	2 povedi	txt
	Proti	2 povedi	txt
	Komentar	1 poved	txt
	Na cesti	10 povedi	txt
	Lastništvo	10 povedi	txt
	Potniški prostor	10 povedi	txt
	Motorna različica	1.8 TDCi	txt
	Podatki o avtomobilu		
	Kategorija	Kompaktna limuzina	txt/končna mnnožica izbora
	Število vrat	4	celo št
	Število sedežev	5	celo št
	Dolžina	4500 [mm]	celo št
	Delovna prostornina	1800 [cm3]	celo št
	Število kW pri vrtljajih	100 [kw]/1400[obratov/min]	celo št
	Parametri za čarovnika		
	Udobje	7,3	zamodel dodana decimalka
	izpusti		
	vodljivost		
	prostornost		
	hitrost in pospeški		
	kvaliteta izdelave		
	stroški		
	Opis		
	Za	2 povedi	
	Proti	2 povedi	
	Komentarji uporabnikov		potreben pogovor
	Seznam dodatne opreme		potreben pogovor

Tabela 1: Seznam potrebnih parametrov

7.3.2 Izpopolnjeni entitetni diagram



Slika 19: Izpopolnjen entitetni diagram

Zaradi dodatnih zahtev so bile dodane naslednje entitete:

- ***cars_group*** – kot vidimo v zgornji tabeli je stranka grupirala dodatne parametre v določene grupe, ker želimo prototip čimbolj prilagoditi zahtevam smo dodali omenjeno tabelo (npr. parametri čarovnika, opis, itd.).

- ***cars_atribute*** – posamezni parameter na tem mestu bomo predvsem zbirali podatke kakšnega tipa bo posamezni atribut (npr. tekst, datum, celo število).

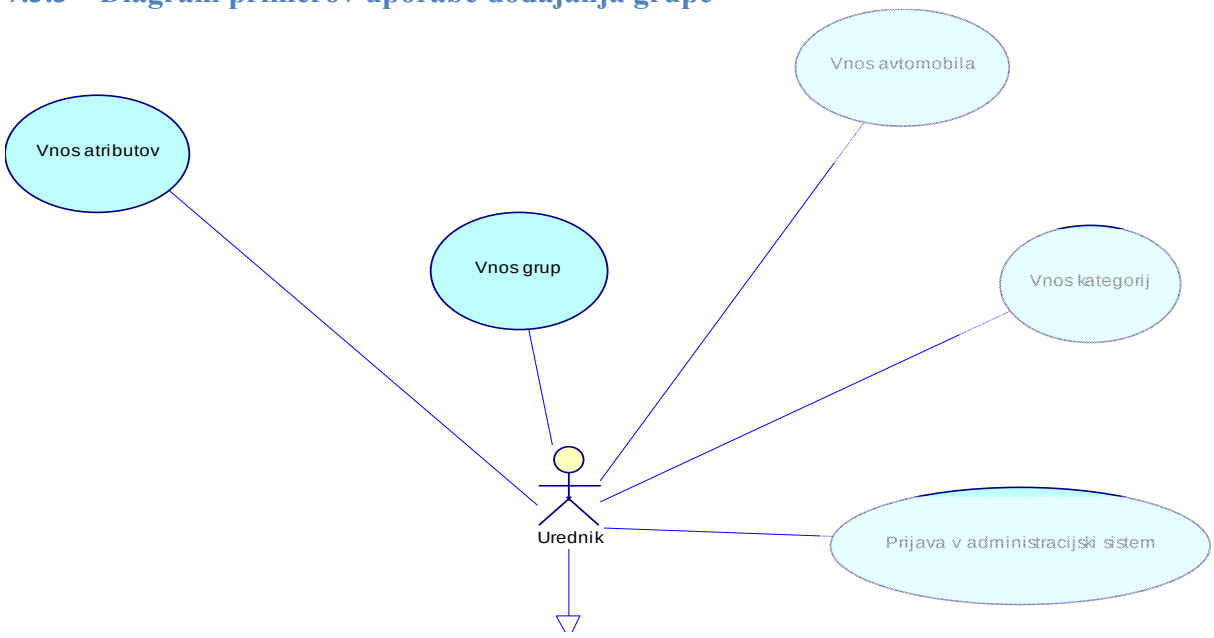
- ***cars_atribute_value*** – ker bodo določeni parametri omejenega nabora, smo na tem mestu umestili tudi to entiteto, zbirali bomo končne vrednosti posameznega atribute (npr. število vrat bo imelo tako že vneseno končno množico vrednosti 2,3,4,5, …, itd.).

- ***cars_groupe_atribute*** – v omenjeni tabeli bomo grupirali atribute na posamezno grupo, dodatno bomo definirali vrstni red posameznega atributa znotraj grupe.

- ***category_group*** – ker ne bodo vse kategorije imele enakih lastnosti smo na tem mestu dodali tabelo *category_group*, tako bomo lahko vsaki kategoriji dodali predhodno definirane grupe, katere se bodo lahko razlikovale od kategorije do kategorije. Dodatno bomo lahko definirali vrstni red posamezne grupe.

- ***category_group_atribute*** – kot je nakazano v zgornji tabeli bodo določene vrednosti vezane na model, drugi pa na različico. Na tem mestu bomo v tabelo *category_group_atribute* vnesli splošne lastnosti posameznega modela, katere se bodo kasneje prenesle na tabelo *car_custom_value*, če bo to urednik zahteval. Na ta način smo omogočili uredniku vnos določenih parametrov, kateri bodo skupni za vse različice modela avtomatsko prenesti na različico, ne da bi za to bilo potrebno dodatno vnašati podatke.

7.3.3 Diagram primerov uporabe dodajanja grupe



Slika 20: Izpopolnjeni diagram primerov uporabe

Zaradi potreb po novih funkcionalnostih, je bilo potrebno posodobiti tudi prvotni diagram primerov uporabe.

7.3.4 Primer uporabe vnos grupe

7.3.4.1 Kratek opis

Vnos grupe, ter definiranje vseh atributov vezanih na posamezno grupo

7.3.4.2 Glavni tok

1. Administrator izbere dodajanje grupe
2. Sistem prikaže vnosno formo s polji:
    a. Naziv grupe
3. Urednik izpolni zahtevane podatke in potrdi vnos
4. Sistem preveri vnose ter shrani zapis v podatkovno bazo
5. Urednik je preusmerjen na zaslonsko masko, kjer iz seznama izbere ustrezne atribute ter te prenese med seznam aktivnih atributov ustrezne grupe
6. Urednik izbere ustrezne atribute in potrdi vnos
7. Sistem preveri izbor ter podatke shrani v podatkovno bazo

7.3.4.3 Alternativni tok

7.3.4.3.1 Dodaj kategorijo

Želenih atributov ni v naboru atributov.

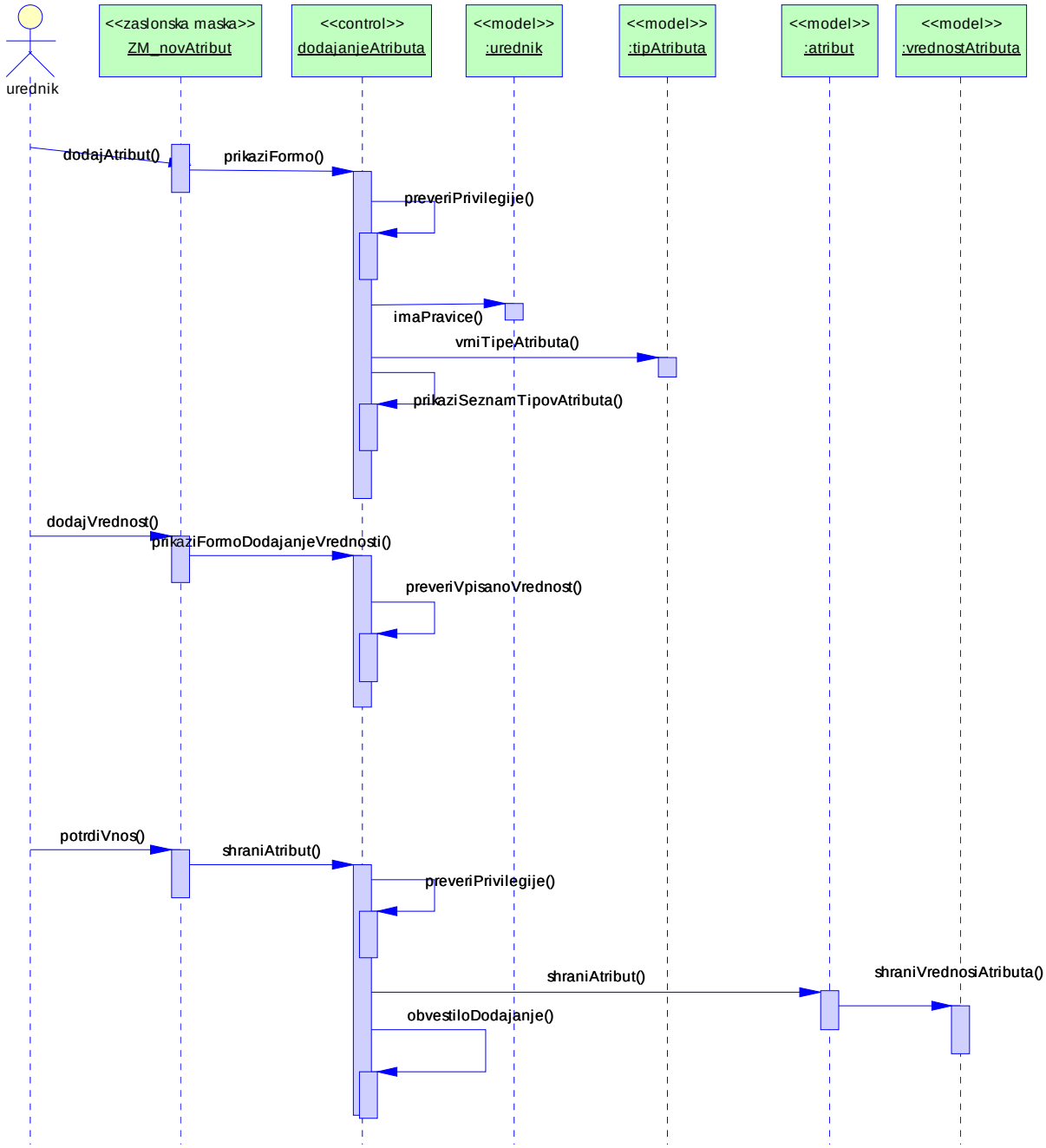
Administrator izbere zaslonsko masko dodajanje nove kategorije.

1. Sistem prikaže trenutno vnešene atribute.
2. Urednik izbere *dodaj atribut*.
3. Sistem pokaže vnosno formo za vnos atributa s polji:
    a. Naziv atributa
    b. Tip atributa
4. Sistem, ob ustrezni izbiri tipa atributa, dodatno, na formi, prikaže naslednja polja:
    a. Vnaprej definirana vrednost atributa
    b. Vnosna polja za vsakega od omogočenih jezikov
    c. Izbira ali je novi vnos neodvisen od jezika (npr. številka vrednost)
5. Sistem preveri vnose ter shrani zapis v podatkovno bazo

7.3.4.4 Posebne zahteve

Urednik, ki dodaja tako grupo kot atribut, mora imeti privilegije nad izbranim modulom. Sistem mora preveriti tip vnesenih podatkov in ga zavrniti v primeru napačnosti. Omogočeno mora biti sortiranje atributov znotraj grupe ter odstranitev.

7.3.5 Diagram zaporedja dodajanje atributa

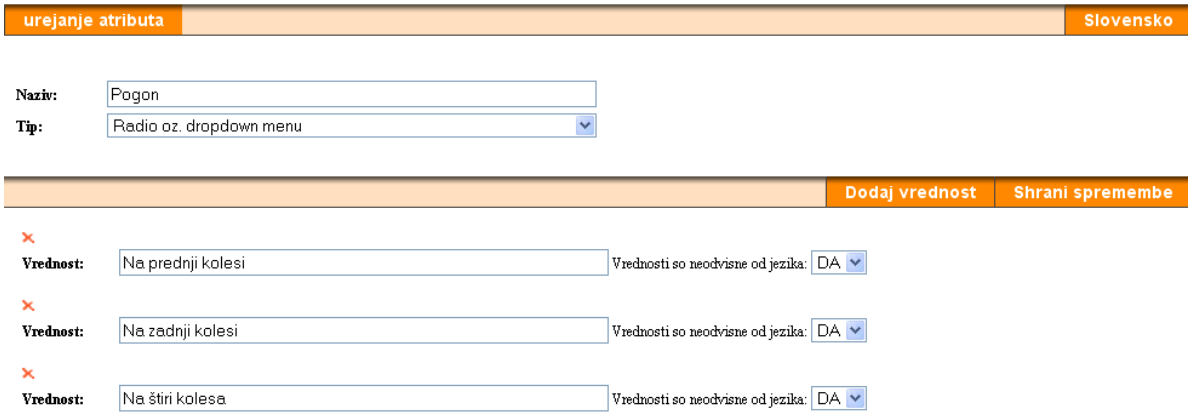


Slika 21: Diagram zaporedja dodajanje atributa

7.3.6 Zaslonska maska dodajanje atributa

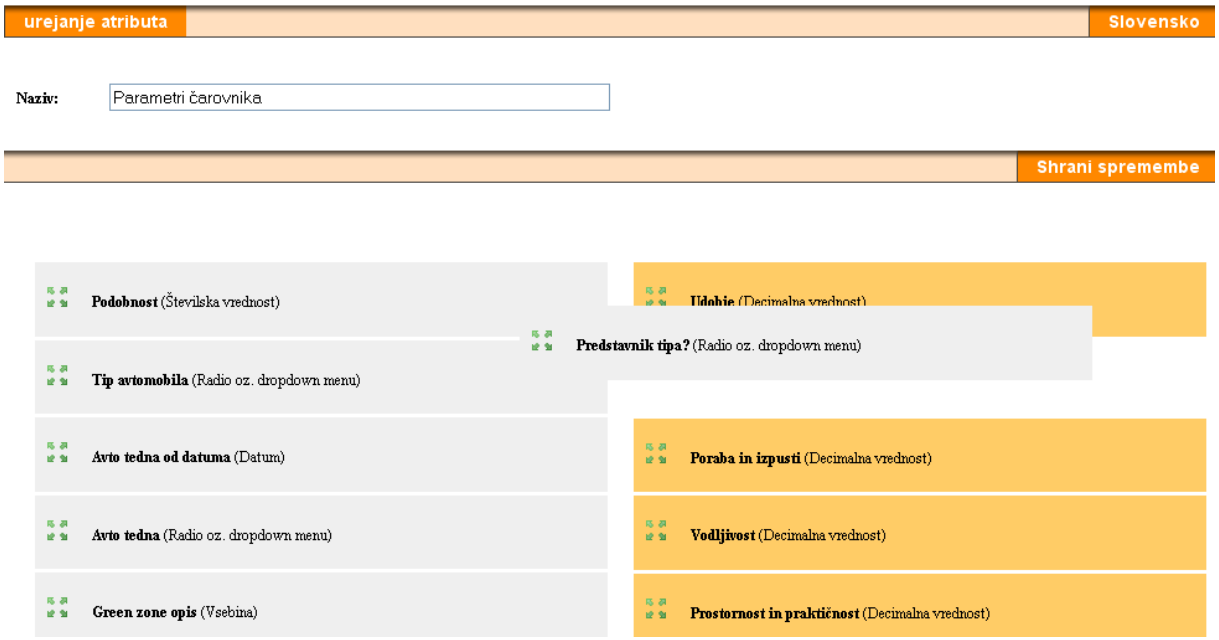
Na zaslonski maski *dodajanje atributa,* smo poskrbeli za definiranje ustreznega tipa atributa. Glede na izbiro se prikaže možnost za dodajanje prednastavljenih vrednosti. Na ta način dosežemo, da pri tipu »*Radio oz. dropdown menu*« urednik vnaprej definira nabor vrednosti pri vnosu kategorije oziroma avtomobila.

Na primeru smo izbrali »*Vrednosti so neodvisne od jezika*«. To pomeni, da bodo ne glede na to kateri jezik bo izbran oziroma kasneje omogočen, vse vrednosti na strani prikazane zgolj v vrednosti, katera je bila izbrana na zaslonski maski.



Slika 22: Zaslonska maska dodajanje atributa

7.3.7 Zaslonska maska dodajanje grupe



Slika 23: Zaslonska maska urejanje grupe

Na tem mestu je bilo potrebno razviti prototip izbiranja in dodajanja atributov k izbrani grupi. V ekipi smo se odločili, da želimo funkcionalnosti čimbolj približati namizni

aplikaciji, zato smo izbrali način premikanja grup s pomočjo prenosa bloka iz leve (neaktivna) na desno (aktivna) stran.

Za ta namen smo s pomočjo *knjižnice jQuery sortable* razvili sistem »primi in spusti« (drag *and drop*) kateri najboljše opiše omenjene funkcionalnosti.

S pomočjo tehnologije *AJAX* smo, nato spremembe, asinhrono posredovali aplikacijskemu nivoju, ta pa je nato shranil zadeve v podatkovno bazo.

Kot lahko vidimo iz spodnje kode, smo razbili levi in desni del na dve grupi. Pri premikanju blokov znotraj vsake oziroma med njima, se pokliče funkcija *update*, ki poskrbi za posodobitev tako na aplikacijskem nivoju (vpis spremembe v podatkovni bazi) kot na predstavitvenem nivoju (prikaz bloka na pravilnem mestu, obarvanje, izpis o spremembi).

```
var leftMenuSize = 0;

jQuery(document).ready(function() {
                jQuery("#groupSortable1, #groupSortable2").sortable({
                                                connectWith: '.connectedSortable',
      handle : '.handle' ,
      update : function () {
            jQuery('#info').fadeIn(2000);
            var order = jQuery('#groupSortable2').sortable('serialize');
            if(order == ""){
               jQuery('#holder2').fadeIn(2000);
            }else{
               jQuery('#holder2').fadeOut(2000);
            }
            jQuery("#info").load("cms.php?show=cars&f=editGroupAtribute&n=1&"+order,
            function (responseText, textStatus, XMLHttpRequest) {
               this; // dom element
               jQuery('#info').fadeOut(2000);
               leftMenuSize = jQuery('#groupSortable1').children().size()-1;
               if(leftMenuSize == 0){
                  jQuery('#holder1').fadeIn(2000);
               }else{
                  jQuery('#holder1').fadeOut(2000);
               }
            });
         }
                                }).disableSelection();
                  });
```

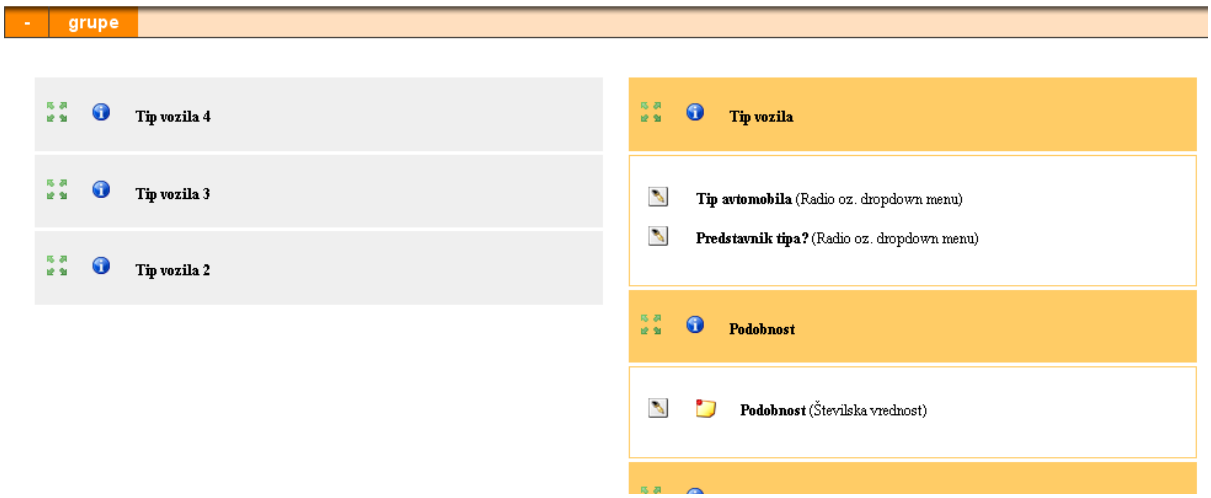
7.3.8 Zaslonska maska dodajanje kategorij

Ker je bilo potrebno sedaj grupe in atribute potegniti na nivo kategorij, je bilo potrebno predhodno razviti prototip nadgraditi z dodatnimi funkcionalnostmi.

Dodajanje grup smo razvili na podoben način kot je bilo to razvito pri dodajanju atributov na nivoju grup.

Potrebno pa je bilo zagotoviti, da se vrednosti sedaj atributom tudi definira. Tako bomo izpolnili zahtevam po potrebnih atributih iz predhodnih dokumentov [Tabela 1].

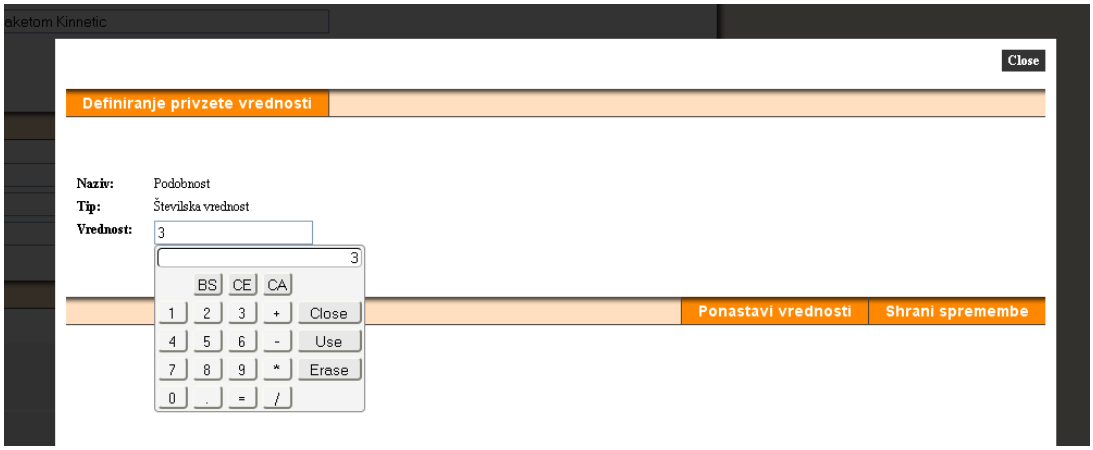
Grupam smo dodali *atribut informacija* (i), ter možnost vnosa vrednosti. Na zaslonski maski dodajanje kategorij vidimo pri grupi *Podrobnost rumeno ikono*, ta nam pove, da smo na tem nivoju atributov *Podobnost* (številska vrednost) že vnesli vrednosti, medtem ko atributu *Tip avtomobila* tega še nismo storili.



Slika 24: Zaslonska maska dodajanje grupe

To lahko storimo tako, da kliknemo na ikono nalivno pero, odpre se nam zaslonska maska urejanje vrednosti atributa, kjer lahko vnesemo zgolj tip vrednosti, katero smo izbrali pri vnosu atributa.

Poskrbljeno je tudi, da se lahko vneseni atributi »resetirajo«, ter da se grupo odstrani iz grupe aktivnih tako, da se jo prenese v levi blok med neaktivne.



Slika 25: Zaslonska maska urejanje vrednosti atributa

7.4 Prototip 1.4

Ker smo v sklopu prototipiranja razvili obsežnejše popravke na *prototipu 1.3*, tako na nivoju predstavitvenega, kontrolnega kot podatkovnega nivoja, smo nove funkcionalnosti opredelili kot novo verzijo prototipa, obstoječe spremembe pa na sestanku predstavili naročniku.

Na ta način smo zbrali mnenja, ter dobili povratne informacije ali smo z nadgradnjami na *prototipu 1.3* ugodili vsem zahtevam.

Ne sestanku so bili definirani naslednji koraki ter nove smernice razvoja spletne aplikacije.

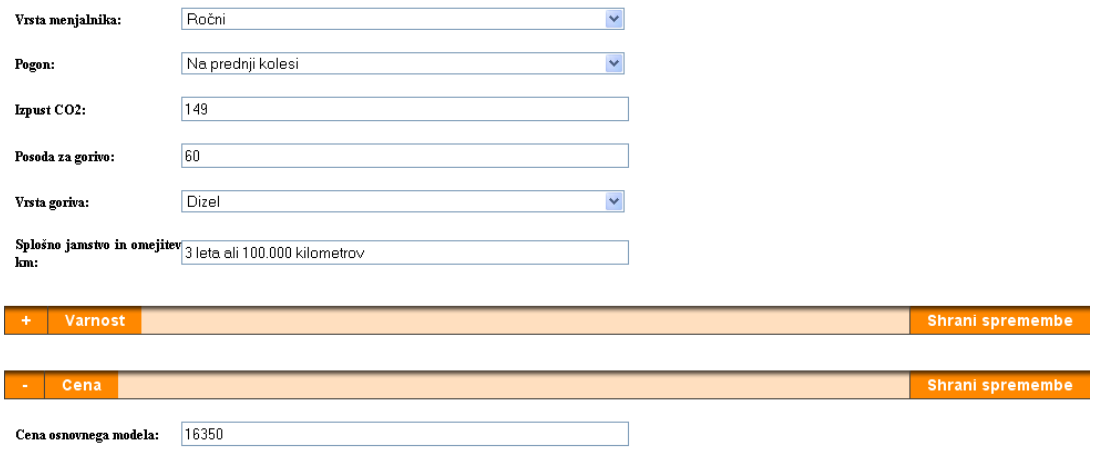
S *prototipom 1.4* želimo zaključiti razvoj administracijskega dela aplikacije, omogočiti urednikom začetek vnosa podatkov, ter preiti na prototip zaslonske maske uporabniškega vmesnika.

7.4.1 Zaslonska maska urejanje avtomobila

V predhodni različici prototipa smo razvili zgolj dodajanje avtomobila, tokrat pa smo nadgradili tako na nivoju kategorij, kot na nivoju avtomobila dodatne parametre v obliki atributov.

Grupe, predhodno dodane na kategorijo, smo sedaj pripeljali na nivo avtomobila. Tako se ob izbiri kategorije avtomobila na zaslonski maski pokažejo dodatni parametri.

Ti so grupirani (npr. varnost, cena, poraba goriva, itd.) znotraj grupe pa imamo atribute (npr. vrsta menjalnika, pogon, cena, itd.).



Slika 26: Zaslonska maska urejanje avtomobila

Dodatno je bilo treba omejiti možnosti vnosa. Tako je treba že pri vnašanju cene poskrbeti da gre za decimalno število. Pri vrsti menjalnika pa smo glede na to, da smo pri vnosu atributa izbrali možnost »*Radio oz. dropdown*« omejili nabor vrednosti na predhodno vnesene.

To je bil tudi naš poglavitni namen. Namreč administracijo bo lahko uporabljala kopica različnih urednikov glede na njihovo hierarhijo, pri tem pa bodo vsi omejeni s strani sistema na vnos, zgolj v naprej definiranih tipov in vrednosti.

7.5 Prototip 2.1

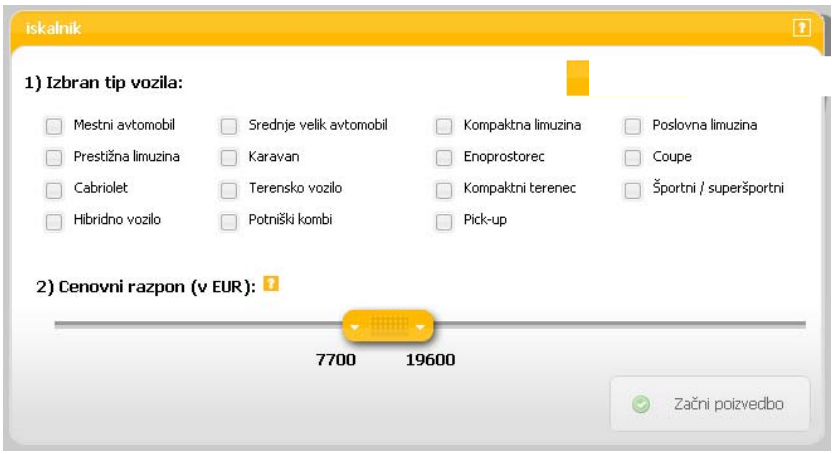
Z izgradnjo funkcionalnosti za dodajanje avtomobilov, smo zagotovili urednikom popolno avtonomijo dodajanja avtomobilov v administracijski sistem.

Sedaj pa je potrebno podpreti podporo prikaza uporabniškega vmesnika obiskovalcem.

Želja je predstaviti iskalnik kot namizno aplikacijo. Skratka želimo doseči čim več interakcije med spletno aplikacijo in uporabnikom s pomočjo miške.

V predhodnih fazah prototipiranja grafičnega vmesnika, smo z naročnikom prišli do končne grafične podobe, bodoče aplikacije. Potrebno je grafiko razrezati, jo najprej pretvoriti v *HTML*, nato pa *HTML* izpopolniti v obliki predlog *Smarty* ter jo na aplikacijskem nivoju podpret z funkcionalnostmi.

V prvi fazi prototipa želimo torej izdelati glavni del bodoče spletne aplikacije. To je iskalnik.



Slika 27: Zaslonska maska naprednega iskalnika

7.5.1 Primer uporabe napredni iskalnik

7.5.1.1 Kratek opis

Iskanje med vnesenimi avtomobili.

7.5.1.2 Glavni tok

1. Obiskovalec izbere tip oziroma več različnih tipov avtomobila
2. Obiskovalec s pomočjo miške premika kurzor na željen cenovni razpon
3. Ob kliku na *začni poizvedbo* aplikacija prikaže avtomobile znotraj iskanega nabora
4. Na novi zaslonski maski lahko uporabnik dodatno filtrira znotraj naslednjih parametrov:
    a. Udobje
    b. Izpusti
    c. Vodljivost
    d. Prostornost
    e. Hitrost in pospeški
    f. Kvaliteta izdelave
    g. Stroški
    h. Varnost
5. Ob spreminjanju se na spletni strani posodabljajo avtomobili glede na izbrane vrednosti parametrov.

7.5.1.3 Alternativni tok

V primeru, da znotraj izbranega nabora ni nobenega vnosa, sistem opozori o tem obiskovalca, ta pa s spremembo parametrov definira nov vnos.

7.5.1.4 Posebne zahteve

Glede na izbiro osmih parametrov je potrebno avtomobile in njihove vrednosti otežiti glede na pomembnost posameznega parametra. Na spletni strani se ne prikazuje zgolj različica avtomobila, temveč model znotraj katerega so predstavljene različne motorne različice posameznega modela. Na seznamu avtomobilov se prikaže tisti model, čigar vsaj ena od različic ustreza izboru parametrov.

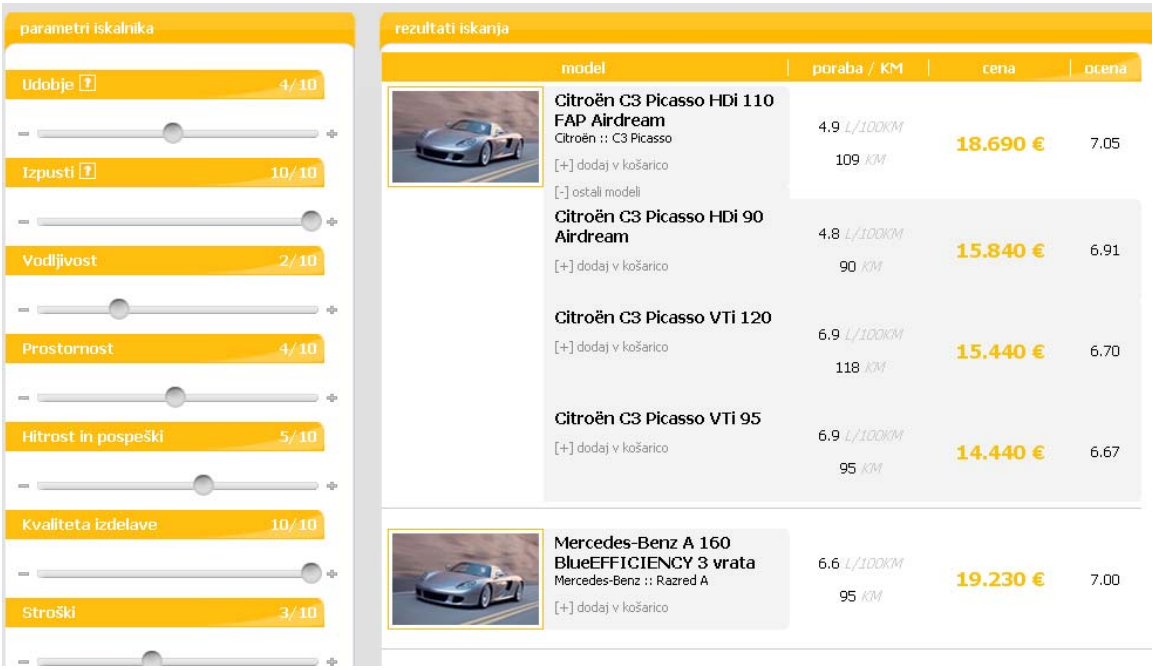
7.5.2 Zaslonska maska napredni iskalnik avtomobilov

Na tem mestu smo razvili zgolj prikaz zaslonske maske v *HTML*. S tem bomo lahko naročniku predstavili končni izgled iskalnika.

Funkcionalnost izbire parametrov (na levi) je sicer že realizirano s pomočjo *knjižnice jQuery*. Prav tako je prikazan izpis modela (*Citroen C3 Picasso*) z vsemi njegovimi

motornimi različicami. Pri tem je različica, ki je po oteženi oceni (ocena) najboljša na prvem mestu, nato pa so ostale motorne različice sortirane po padajoči oceni.

Dodali smo tudi opcijo »*dodaj v košarico*«, katero bomo v kasnejših prototipih nadgradili do nivoja primerjave več različnih modelov.



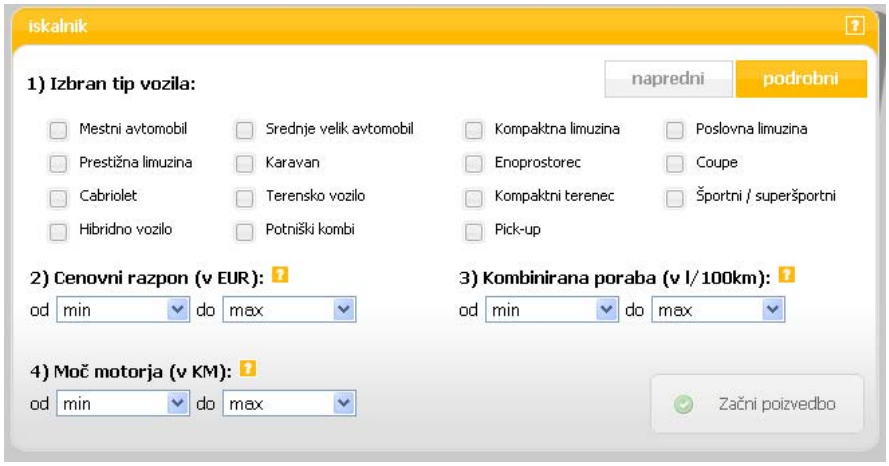
Slika 28: Zaslonska maska naprednega iskalnika avtomobilov

7.6 Prototip 2.2

Ker ne želimo zapravljati časa po nepotrebnem, smo takoj, ko je bil grafični vmesnik razvit v delno delujoč prototip vmesnika v kodo *HTML,* predstavili naročniku.

Tako je lahko naročnik na konkretnem primeru preveril delovanje bodočega sistema. Z določenimi popravki je bil prototip potrjen. Določen je bil rok za funkcionalno podprtje aplikacijskega nivoja.

Ugotovljena je bila potreba po dodatnem »podrobnem« iskalniku. Ta bi bil kot dodatni iskalnik namenjen vsem uporabnikom, ki se bolje spoznajo v vrednosti parametrov in ne potrebujejo vnaprej definiranih vrednosti.

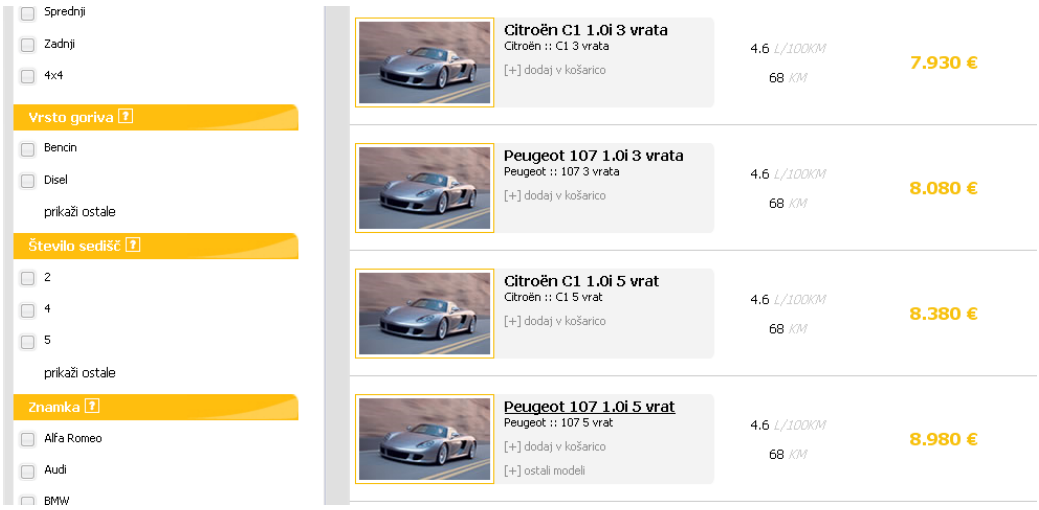


Slika 29: Zaslonska maska podrobnega iskalnika

Cenovni razpon je bil sedaj definiran ročno. Dodana pa sta bila dodatna parametra kombinirana poraba in moč motorja.

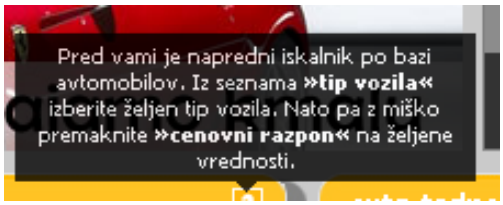
Potrebni so bili tudi popravki pri dodatnih parametrih.

Tako je prejšnjih 8 parametrov, sedaj zamenjalo 5 parametrov. Kateri so pa ročno izbrani glede na specifične zahteve uporabnikov.



Slika 30: Zaslonska maska podrobnega iskalnika avtomobilov

Dodatno so bile dodane pomoči v obliki vprašajev na zaslonskih maskah, ter tako dosegli dodatno preglednost in razlago za začetnike.

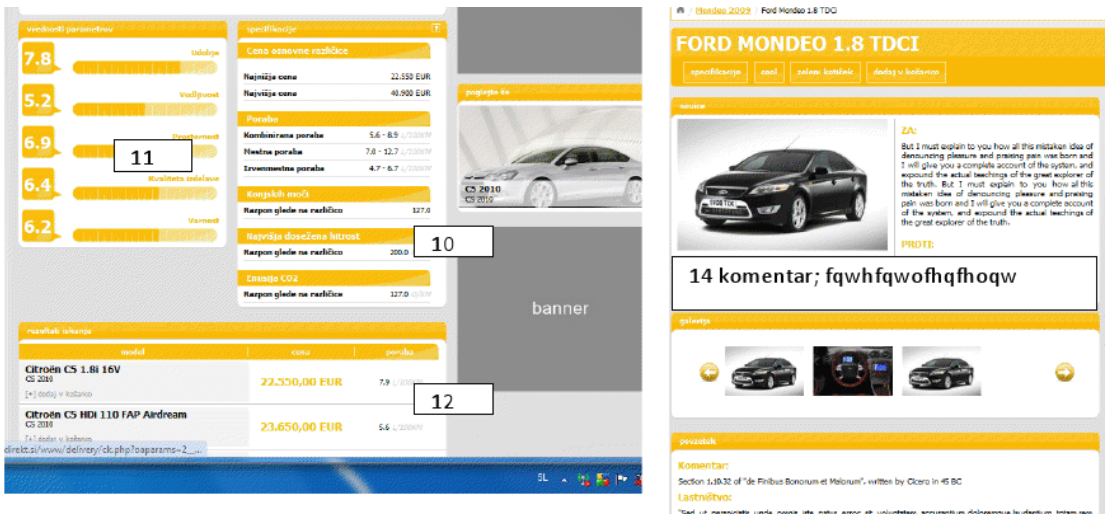


Slika 31: Zaslonska maska pomoči

7.7 Končni sistem

V predhodnih prototipih smo naročniku predstavili bodoče funkcionalnosti. S popravki na *prototipu 2.2,* pa smo vse predhodno definirane funkcionalnosti tudi implementirali v končni sistem.

Od naročnika smo prejeli na zadnjem sestanku dodatne pripombe glede pomanjkljivosti predhodnih prototipov, katere se bodo do končne verzije popravile.



Slika 32: Primer zahtevanih popravkov

12	pri pregledu modela naj bodo pod porabo tudi konji
13	pri pregledu konkretne motorne različice naj bo pri specifikacijah namesto številk pokaži vse
14	komentar naj se pojavi pod sliko avta, ne v opisu
15	pod naslov *videa*, novice, nasveta, se doda datum in naj bo tudi, ko odpreš npr. Novico

Tabela 2: Seznam končnih popravkov na prototipu

8 Zaključek

Pri ocenjevanju primernosti posamezne metodologije razvoja informacijskega sistema, je vedno potrebno v razmislek vzeti tudi posamezne parametre projekta (npr. obseg projekta, določeno pogodbeno razmerje med kupcem in izvajalcem, spreminjanje zahtev). To so le nekatera vprašanja, ki vplivajo na izbiro pravilne metodologije.

Pri razvoju spletnih aplikacij je smiselno uporabiti prototipni pristop, kadar imamo velik obseg modifikacij sistema, dodatnih programskih rešitev, ki jih je potrebno razviti, kadar zahteve niso jasno definirane, kadar je razvojna ekipa relativno majhna. Ti dejavniki so prisotni pri večini problemskih domen.

Prototipiranje, skozi iteracijski cikel razvoja, ponuja nenehno prilagajanje in usklajevanje, vse to z namenom, da bi čim bolje razumeli in zadostili namenu končnega informacijskega sistema.

Največja težava prototipnega pristopa utegne biti definicija pogodbenega razmerja. Za uspešno uporabo njegovih prvin je v veliki meri potrebno ne le precejšnje sodelovanje naročnika, temveč tudi njegovo razumevanje problematike ter okoliščin v katerih lahko izkoristimo prednosti takega razvoja.

Tako je ravno največji izziv, ki ostaja, pravilna in uspešna predstavitev procesa.

9 Literatura

[1] B. Hunter, M. Fowler, G. Hohpe, Agile EAI Methods: Minimizing Risk, Maximizing ROI, ThoughtWorks Inc., Julij 2002

[2] S. Chatterhee, Managing EAI Projects in Agile way: road towards a successful EAI implementation, Cap Gemini Ernst & Young Consulting India Pvt. Ltd., 2004

[3] Manifesto for Agile Software Development (http://agilemanifesto.org/)

[4] M. Fowler, G. Hohpe, Agile EAI, ThoughtWorks, November 2002

[5] G. Hohpe, W. Istvanick, Test-Diven Dvelopment in Enterprise Integration Projects, ThoughWorkd, November 2002

[6] P. Abrahamsson, Agile software development methods: Review and analysis, VTT Electronics, 2002

[7] John Crinnion: Evolutionary Systems Development, a practical guide to the use of prototyping within a structured systems methodology. Plenum Press, New York, 1991. Page 18.

[8] S. P. Overmyer: Revolutionary vs. Evolutionary Rapid Prototyping: Balancing Software Productivity and HCI Design Concerns. Center of Excellence in Command, Control, Communications and Intelligence (C3I), George Mason University, 4400 University Drive, Fairfax, Virginia.

[9] Software Productivity Consortium: Evolutionary Rapid Development. SPC document SPC-97057-CMC, version 01.00.04, June 1997. Herndon, Va. Page 6.

[10] Davis. Page 72-73. Citing: E. Bersoff and A. Davis, Impacts of Life Cycle Models of Software Configuration Management. Comm. ACM, Aug. 1991, pp. 104–118

[11] Joseph E. Urban: Software Prototyping and Requirements Engineering. Rome Laboratory, Rome, NY.

[12] (2010) Dynamic Systems Development Method Consortium. Dostopno na:

http://na.dsdm.org

[13] Alan M. Davis: Operational Prototyping: A new Development Approach. IEEE Software, September 1992. Page 71.

[14] Paul W. Parry. Rapid Software Prototyping. Sheffield Hallam University, Sheffield, UK.

[15] Dr. Ramon Acosta, Carla Burns, William Rzepka, and James Sidoran. Applying Rapid Prototyping Techniques in the Requirements Engineering Environment. IEEE, 1994.

[16] Top 10 Simulation Tools for UI Designers, Information Architects and Usability Specialists. Dostopno na:

http://uidesign-usability.blogspot.com/2007/03/top-10-simulation-tools-for-ui.html

[17] C. Melissa Mcclendon, Larry Regot, Gerri Akers: The Analysis and Prototyping of Effective Graphical User Interfaces. October 1996

[18] Cockburn, A., »Agile Software Developement«, Pearson Education Inc., 2002

[19] (2010) Iterative and incremental development. Dostopno na:

http://en.wikipedia.org/wiki/Iterative_and_incremental_development

[20] Highsmith J., Cockburn A: »Agile software developement ecosystems«, Addison-Wesley, 2002

[21] (2010) Systems Development Life Cycle. Dostopno na:

http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle

[22] (2010) How Web Site Prototyping Can Build Better Client-Provider Relationships. Dostopno na:

http://www.articlecompilation.com/Article/How-Web-Site-Prototyping-Can-Build-Better-Client-Provider-Relationships/540843

[23] (2010) Agilne metode razvoja programske opreme. Dostopno na:

http://sl.wikipedia.org/wiki/Agilne_metode_razvoja_programske_opreme

[24] (2010) ISO 12207. Dostopno na:

http://en.wikipedia.org/wiki/Software_Lifecycle_Processes

[25] (2010) JQuery Usage Statistics. Dostopno na:

http://trends.builtwith.com/javascript/JQuery

[26] (2010) Multitier architecture. Dostopno na:

http://en.wikipedia.org/wiki/Multitier_architecture