# RAZVOJ PROGRAMSKE OPREME Z UPORABO JEZIKA UML

Aleš Živkovič, Marjan Heričko, Ivan Rozman
Fakulteta za elektrotehniko, računalništvo in informatiko
Inštitut za informatiko
e-pošta: {ales.zivkovic, marjan.hericko, i.rozman}@uni-mb.si

**Izvleček**

V prispevku je predstavljen jezik za dokumentiranje dela in rezultatov pri objektnem razvoju programske opreme. Predstavili bomo zgradbo jezika UML, diagramske tehnike in dodatne izrazne možnosti, vpeljane v obliki dopolnitev k osnovnim gradnikom jezika. Zelo natančen in dobro zasnovan standard pa ne zajema napotkov za uporabo jezika, zato potrebujemo še proces. Spoznali bomo, da univerzalnega procesa ni, prilagoditi ga moramo okolju, v katerem se uporablja, in ga nenehno izboljševati. Podali bomo priporočila za oblikovanje procesa razvoja programske opreme, kadar sledimo objektni paradigmi.

***Abstract***

*In the paper language for documenting software development artifacts will be presented. First the structure, diagram elements and supplemental expressive possibilities introduced as an extension mechanisms for basic techniques will be described. Unfortunately, well-formed standard doesn't define when and how to use its elements. Therefore we need the software development process to define steps, inputs and outputs for each step. There is no single process suitable for all the environments. Process needs to be tailored to environment specifics and continuously improved. Recommendations for forming an object-oriented software development process will also be presented.*

■ ■ ■

## 1. Uvod

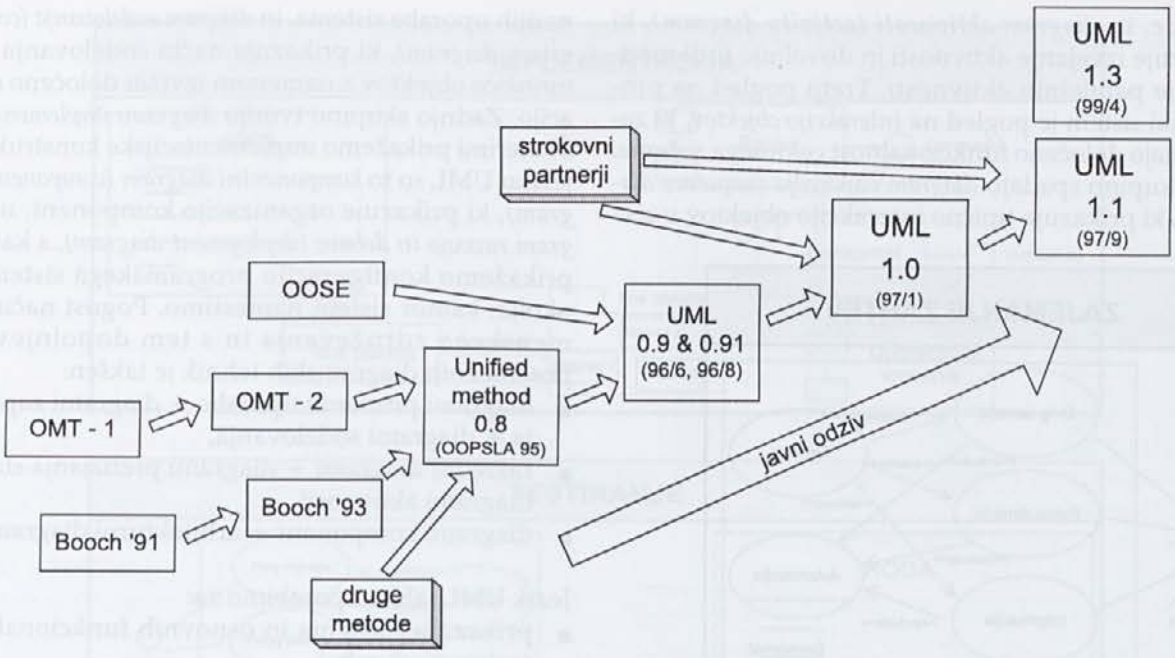
Kljub vedno boljšim integriranim okoljem za razvoj programske opreme, ki nam omogočajo enostavno delo in abstrakcijo kode, ki nastaja, je potreba po modeliranju in načrtovanju programske opreme neizmerna. Tehnološki koncepti, ki vodijo do učinkovitih rešitev, so postali kompleksni. Direkten preskok v implementacijo, brez jasno začrtane poti, je praktično nemogoč, če pa že, pa je zagotovo neučinkovit in časovno potraten. Načrte potrebujemo v takšni ali drugačni obliki. Lahko jih oblikujemo čisto samoiniciativno in na svoj način ali pa sledimo natančno določenim postopkom in korakom. Veliko je odvisno od skupine, tipa in velikosti projekta in nenazadnje organizacije in vodstva. Ne glede na vse to, se razvijalci zavedamo prednosti uporabe posameznih diagramskih tehnik, ki v nobenem pogledu niso zgolj lepe slikice. So način komuniciranja na višjem, bolj razumljivem nivoju abstrakcije in s tem neodvisni od izbranega programskega jezika. Zaradi množice notacij, ki so bile v preteklosti v uporabi, je bila komunikacija med razvijalci otežena in je pogosto povzročala zmedo. Po letu 1996 so različne notacije konvergirale v en sam, standarden nabor diagramskih tehnik, prvič predstavljen na konferenci OOPSLA'95 [3]. Kasneje so pobudo podprla številna podjetja, kot tudi skupina OMG[5]. Vpeljano je bilo ime Unified Modelling Language (UML) [5], ki še bolj poudarja, da proces v njem ni zajet. Dobili smo standarden jezik za dokumentiranje postopkov in rezultatov pri razvoju programske opreme. Značilnosti lahko strnemo v naslednjih točkah:

- je vizualno orodje za modeliranje,
- ima standardizirano notacijo,
- omogoča zajemanje poslovnega procesa,
- izboljša komunikacijo,
- omogoča obvladovati kompleksnost,
- definira arhitekturo programske opreme,
- pospešuje ponovno uporabo,
- omogoča uporabo poljubnega razvojnega procesa.

Razvoj jezika prikazuje slika 1.

## 2. Jezik UML

Jezik UML definira notacijo in metamodel jezika. Notacija je grafična predstavitev jezika, sintaksa, ki sama po sebi ne daje jasnega in nedvoumnega pomena posameznih konstruktov. Splošna uporaba notacije




**Slika 1: Zgodovinski razvoj jezika UML**

največkrat definira nekatere neformalne definicije, toda bolje bi bilo poiskati bolj formalno definicijo. Ideja rigoriznih specifikacij in jezikov načrtovanja prevladuje pri uporabi formalnih metod. Kadar uporabljamo tehnike s tega področja, izražamo specifikacije in načrtovanje s pomočjo izpeljank predikatnega računa. Takšne definicije so matematično natančne in ne dopuščajo dvoumnosti, a na žalost niso vsesplošno uporabne. Tudi kadar lahko dokažemo, da program zadovoljuje matematično specifikacijo, nikakor ne moremo dokazati, da te specifikacije v resnici izpolnjujejo uporabnikova pričakovanja.

Avtorji jezika za objektno modeliranje UML so jeziku zagotovili formalno podlago, ne da bi žrtvovali njegovo uporabnost. Ena od možnosti je definicija metamodela – največkrat razrednega diagrama, ki definira notacijo in natančno ter nedvoumno definira pomen posameznih gradnikov. Avtorji so jezik za objektno modeliranje UML zgradili štiriplastno:

- *Meta–metamodel*, ki definira jezik za specifikacijo metamodela in predstavlja infrastrukturo za arhitekturo metamodela.
- *Metamodel*, ki je primerek meta-metamodela in definira jezik za specifikacijo modelov.
- *Model*, ki je primerek metamodela in definira jezik za opis informacijske domene.
- *Uporabniški objekti*, ki so primerki modela in definirajo konkretno informacijsko domeno.

Takšna zasnova jezika omogoča uporabo jezika na vseh področjih, saj vsebuje gradnike, ki omogočajo modeliranje različnih sistemov. Poleg formalne zasnove jezika so avtorji dodali tudi jezik OCL (*Object Constraint Language*) – jezik za določanje omejitev, ki je popolnoma formalen in omogoča oblikovanje omejitev vseh modelov (vloge v asociacijah, številnost povezav, pred- in po-pogoji metod, ipd.). Kot kažejo izkušnje, je kombinacija formalne zasnove jezika za modeliranje prinesla prednosti na dveh področjih. Po eni strani je jezik splošno uporaben, a jasno definiran, kar omogoča uporabo jezika UML na vseh problemskih področjih, po drugi strani pa jasno definirana struktura jezika omogoča lažji razvoj podporne programske opreme.

## 2.1 Gradniki

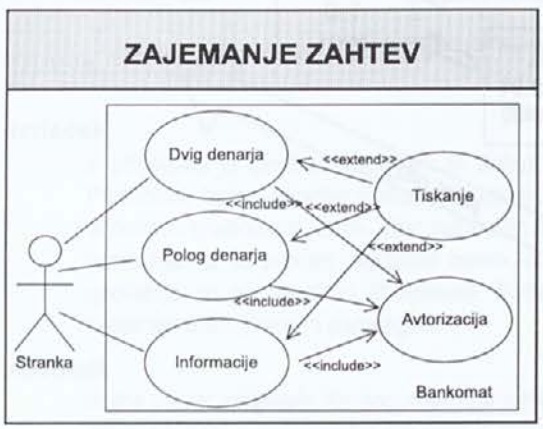
Iz diagramskih tehnik jezika UML lahko razberemo štiri različne poglede na problemsko področje. Prvi je pogled na uporabniške zahteve, kjer uporabljamo *diagram primerov uporabe (use case diagram)*, ki izvira neposredno iz Jacobsonove metodologije in je bil za potrebe jezika UML poenostavljen. V drugo skupino lahko uvrstimo *razredni diagram (class diagram)*, ki prikazuje statično sliko sistema. Razredni diagram je sestavljen iz razredov in povezav med razredi in je le nekoliko spremenjena diagramska tehnika iz metodologije OMT (*Object Modeling Technique, Rumbaugh*)[1]. Tretji pogled predstavlja skupina diagramskih tehnik, ki modelirajo *obnašanje programskega sistema*. Tu najdemo *diagram prehajanja stanj (statechart diagram)*, s katerim opisujemo prehajanja stanj za posamezne

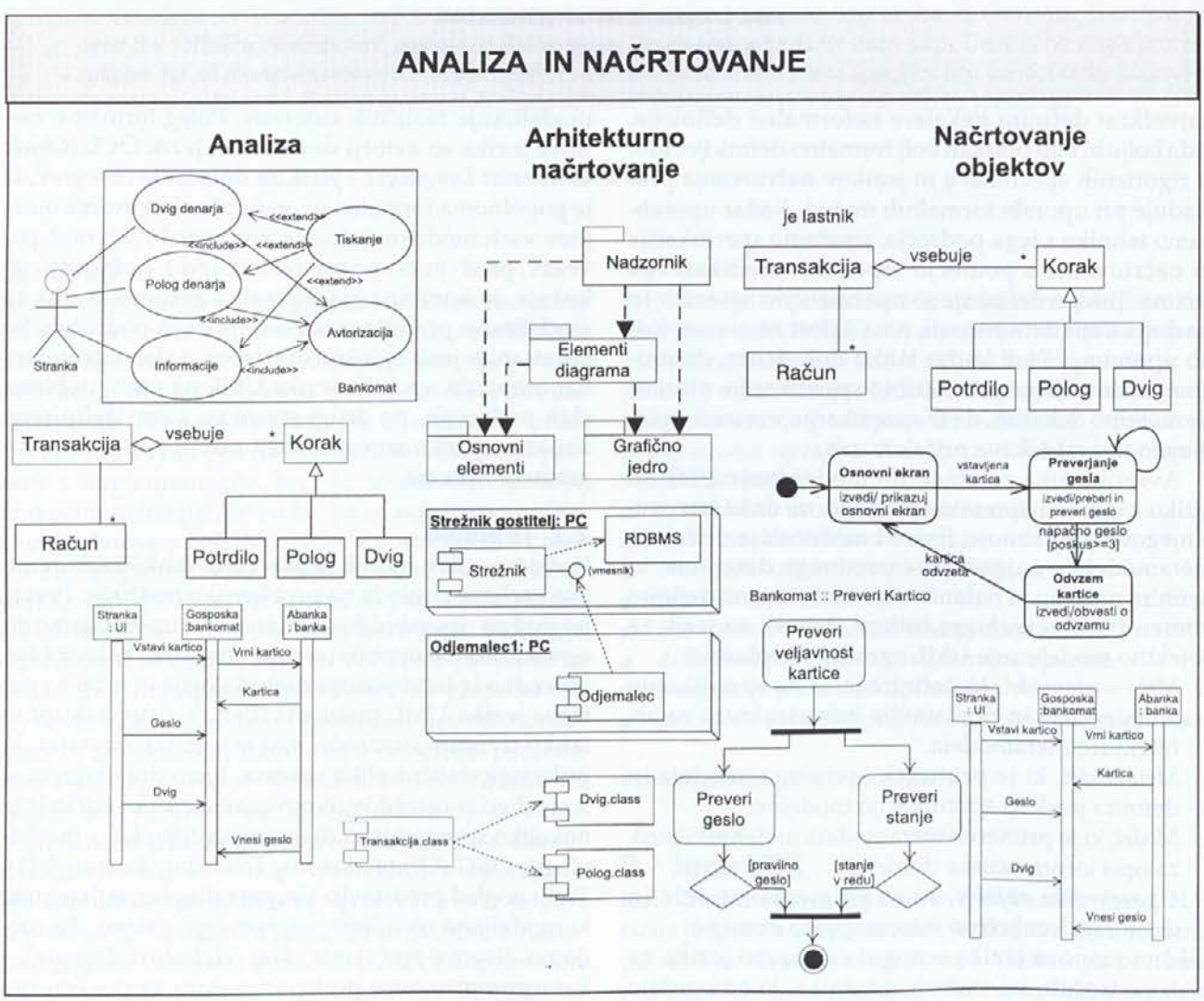
razrede, in *diagram aktivnosti (activity diagram)*, ki prikazuje izvajanje aktivnosti in dovoljuje tudi modeliranje paralelnih aktivnosti. Tretji pogled na programski sistem je pogled na *interakcijo objektov*, ki zagotavljajo določeno funkcionalnost celotnega sistema. V to skupino spadajo *diagram zaporedja (sequence diagram)*, ki prikazuje tipično interakcijo objektov v scenarijih uporabe sistema, in *diagram sodelovanja (collaboration diagram)*, ki prikazuje način sodelovanja med množico objektov z namenom izvršiti določeno operacijo. Zadnjo skupino tvorijo *diagrami implementacije*, s katerimi prikažemo implementacijske konstrukte. V jeziku UML so to *komponentni diagram (component diagram)*, ki prikazuje organizacijo komponent, in *diagram razvoja in dobave (deployment diagram)*, s katerim prikažemo konfiguracijo programskega sistema in okolja, kamor sistem namestimo. Pogost način pomenskega združevanja in s tem dopolnjevanja posameznih diagramskih tehnik je takšen:

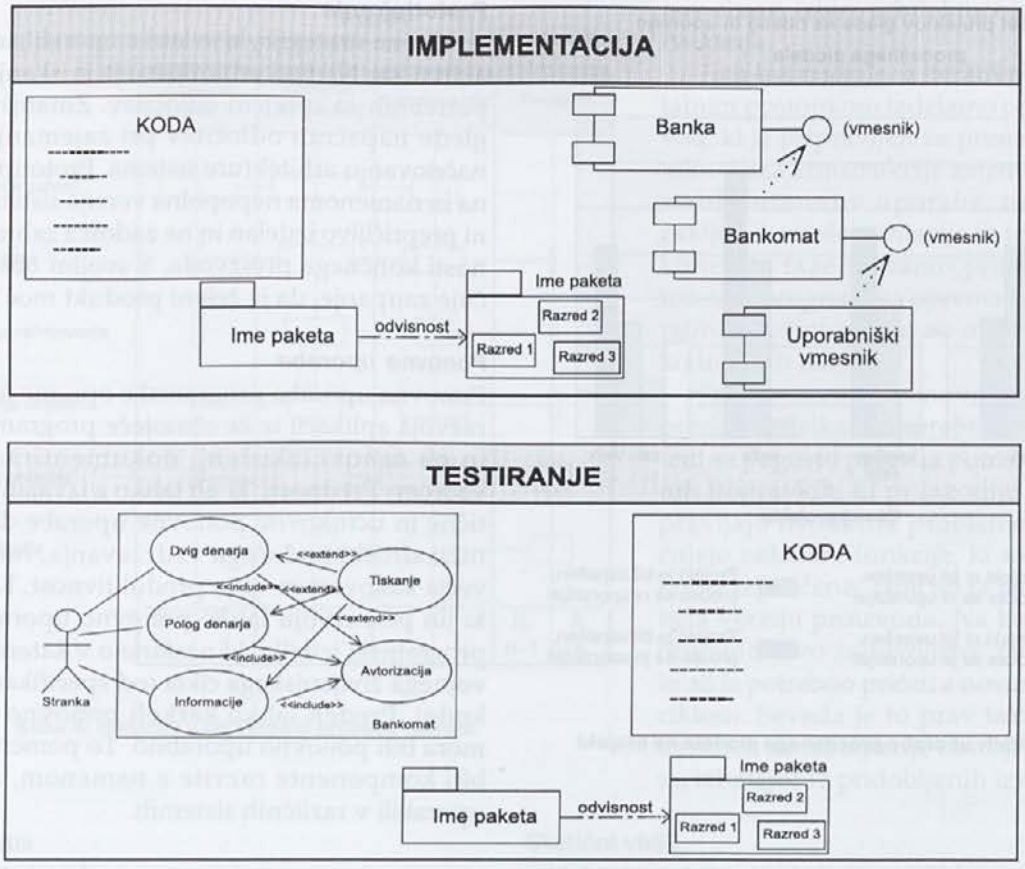
- diagrami primerov uporabe + diagrami zaporedja + diagrami sodelovanja,
- razredni diagrami + diagrami prehajanja stanj + diagrami aktivnosti,
- diagrami komponent + arhitekturni diagrami.

Jezik UML lahko uporabimo za:

- prikaz mej sistema in osnovnih funkcionalnosti (primeri uporabe),






**Slika 2: Shematski prikaz uporabe diagramskih tehnik**

- prikaz realizacije sistema (diagrami interakcije),
- prikaz statične strukture sistema (razredni diagrami),
- modeliranje obnašanja objektov (diagrami stanj),
- določitev fizične implementacije arhitekture (arhitekturni diagrami).

Osnovne gradnike lahko dopolnimo z naslednjimi koncepti:

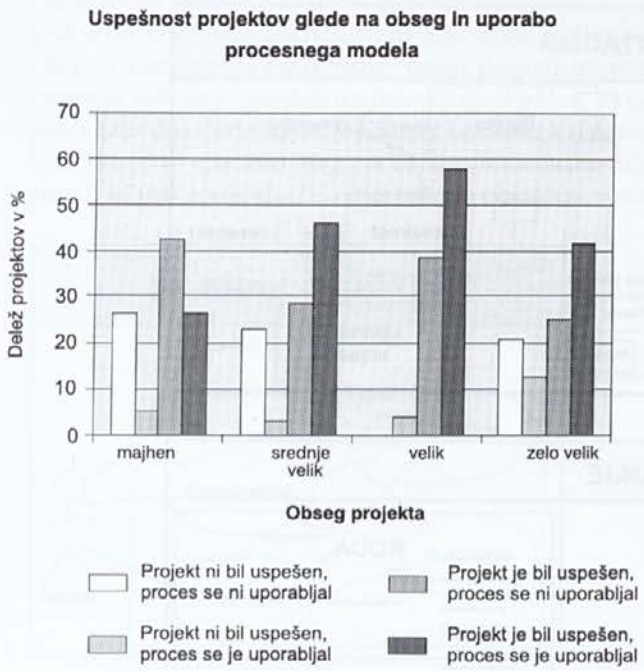
- *Stereotip* (stereotype) - poljubnemu elementu jezika damo nov pomen, oziroma pomen prilagodimo zahtevam razvojnega procesa.
- *Imenovana vrednost* (tagged values) - par oznaka, vrednost lahko dodamo kateremukoli elementu modela. Nekatera imena so že definirana.
- *Omejitve* (constraints) - diagram lahko natančneje opišemo in dodamo natančne pogoje kdaj in pod kakšnimi pogoji se bo nekaj zgodilo.

## 3. Razvojni proces

### 3.1 Vpliv procesnega modela na projekt

Iz primerjave vplivov različnih dejavnikov na projekt [7] lahko posredno sklepamo, da je urejenost postopkov in ponovljivost pri izvajanju projektov prav tako pomemben dejavnik, saj je zrelost organizacije bodisi po Zrelostno zmožnostnem modelu (Capability Maturity Model CMM) ali skladno s standardom 9000-3 [6] označena kot tretji najbolj pomemben dejavnik, takoj za človeškim faktorjem in pretokom informacij med vpletenimi v projekt. S poznavanjem CMM in ISO 9000-3 lahko ugotovimo posredno pomembnost uporabe procesnega modela za razvoj programske opreme. Predpostavko potrjuje povezava med uspešnostjo, obsegom projekta in uporabo procesnega modela. Iz slike 3 lahko razberemo, da je uporaba natančno določenih postopkov pri razvoju programske opreme bolj pomembna pri večjih projektih, kjer je odstotek uspešno zaključenih projektov bistveno večji v primerjavi s projekti, kjer se procesni model ni uporabljal.

Povedali smo že, da je jezik UML zgolj notacija in ne metoda razvoja, saj nima definiranega procesa razvoja, ki je pomemben del metode. Avtorji so že na samem začetku ugotovili, da splošnega procesa razvoja ne bo moč razviti, zato predlagajo uporabo ogrodja, kjer si vsak uporabnik proces lahko prilagodi svojim zahtevam in uporabi tiste diagramske tehnike, ki so primerne in smiselne za njegov razvojni proces.




**Slika 3: Vpliv uporabe procesnega modela na projekt**

### 3.2 Značilnost objektnega procesnega modela

Predlagamo, da objektni proces razvoja IS združuje naslednje pomembne strategije:

#### *Iterativni razvoj*

Za mnoge razvijalce programske opreme je uporaba iteracij ena izmed ključnih značilnosti njihovega dela. Praktično to pomeni odlašanje izvedbe za določeno časovno obdobje z namenom hitrejšega napredovanja in kasnejšega vračanja k istemu problemu, kjer dosežene rezultate izboljšamo oziroma popravimo. S takšnim načinom dela se lažje prebijemo skozi kritične točke dela.

#### *Inkrementalni razvoj*

Inkrement pomeni dodatek k nečemu - napredovanje. Inkrement pomeni korak bliže k izpolnitvi zadanih ciljev. Inkrementalni razvoj je strategija napredovanja v malih korakih, da bi prišli do ustreznih rezultatov. Inkrementalni pristop zahteva razdelitev problema na podprobleme tako, da jih lahko rešujemo sočasno in izmenično. Ob rešitvi vsakega podproblema le-tega testiramo in povežemo z ostalimi deli sistema. Izraza iterativni in inkrementalni se pogosto uporabljata v navezi ali izmenično in opisujeta uporabljen procesni model pri razvoju predvsem objektnih aplikacij. Strategiji sta med seboj ločeni in neodvisni.

#### *Prototipiranje*

Gre za strategijo, ki jo lahko uporabljamo pri večini aktivnosti. Namen prototipiranja je iskanje informacij, potrebnih za sprejem odločitev. Zmanjšuje tveganje glede napačnih odločitev pri zajemanju zahtev in načrtovanju arhitekture sistema. Prototip je predhodna in namenoma nepopolna verzija sistema. Običajno ni prepričljivo izdelan in ne zadošča zahtevam robustnosti končnega proizvoda. S svojim obstojem povečuje zaupanje, da je želeni produkt moč izdelati.

#### *Ponovna uporaba*

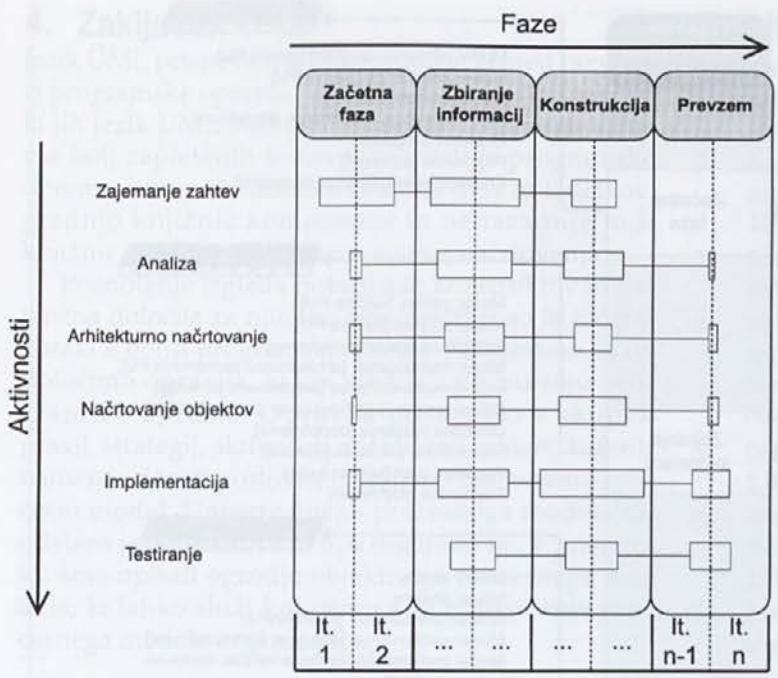
Ponovna uporaba programske opreme je sposobnost razvoja aplikacij iz že obstoječe programske opreme in na osnovi izkušenj, dokumentiranih v obliki vzorcev. Prednosti, ki jih lahko z izvajanjem sistematične in učinkovite ponovne uporabe dosežemo, so nižji stroški bodočega vzdrževanja, hitrejši razvoj, večja kakovost in višja produktivnost. Komponente, ki jih pri razvoju lahko ponovno uporabimo, so vsi programski izdelki, ki nastanejo v katerikoli fazi razvojnega življenjskega cikla (od specifikacij zahtev do kode). Preden lahko karkoli ponovno uporabimo, mora biti ponovno uporabno. To pomeni, da morajo biti komponente razvite z namenom, da jih bomo uporabili v različnih sistemih.

### 3.3 Zgradba

Proces razvoja programske opreme je razdeljen na posamezne faze. Pri objektnem razvoju IS ločimo štiri osnovne faze (*začetna faza, zbiranje zahtev, konstrukcija, prevzem*). Faze prikazujejo časovno delitev objektnega procesa. Iterativno inkrementalna narava objektnega pristopa narekuje predstavitev procesnega modela v dveh dimenzijah:

- po času – predstavlja življenjski cikel procesa,
- po aktivnostih – predstavlja sestavne dele procesa.

Tovrstna predstavitev v procesnem modelu poveže dva do sedaj ločena pogleda na razvoj informacijskih sistemov. Gledano iz časovne perspektive govorimo o fazah, katere delimo na razvojne cikle - iteracije in mejnike. Tak pogled je značilen za upravljalski nivo gledanja na projekt (projektno vodenje). Le-ta zajema časovno komponento, sredstva, ljudi in organizacijo dela. Gledano iz perspektive proizvodov pa razvojni proces delimo na posamezne aktivnosti. Pri tem ločimo tehnične aktivnosti in podporne aktivnosti (*upravljanje konfiguracij in sprememb, vodenje projektov, uporaba metrik, upravljanje okolja*). Slika 4 prikazuje le tehnične aktivnosti. Iz opisa objektnega procesa razvoja IS so izvzete tudi aktivnosti, ki se ne spremenijo z objektnim pristopom, npr. načrtovanje grafičnega uporabniškega vmesnika, oblikovanje uporabniške dokumentacije, oblikovanje tehnične dokumentacije.




**Slika 4: Iteracije v objektnem procesu razvoja**

gov obseg, izbiro arhitekture in morebitna tveganja.

*Faza konstrukcije* - z iterativno inkrementalnim postopkom izdelamo celoten proizvod, ki je pripravljen za prenos k uporabniku. Faza konstrukcije zajema opis preostalih primerov uporabe, načrtovanje, zaključek implementacije in testiranje. Na koncu te faze moramo presoditi, ali so izdelani programska oprema kot tudi uporabniki, pripravljeni za opravljanje vsakodnevnih nalog.

*Faza prevzema* - osnovni cilj te faze je predaja izdelka v uporabnikovo okolje. S tem se pogosto pojavlja potreba po dodatnih popravkih, ki prilagodijo sistem, popravljajo neodkrite probleme ali dokončujejo nekatere funkcije, ki so bile namenoma izpuščene. Tipično ta faza nastopi z beta verzijo proizvoda. Na koncu te faze ocenimo nivo zadovoljitve zadanih ciljev in ali je potrebno pričeti z novim razvojnim ciklom. Seveda je to prav tako primeren trenutek za izboljšanje obstoječega procesa, izhajajoč iz pridobljenih izkušenj.

### Dinamični vidik

Življenjski cikel vsake generacije[1] programske opreme delimo na štiri faze. Vsaka faza se zaključi z definiranim časovnim mejnikom in zahteva sprejem odločitev, ki lahko v veliki meri vplivajo na uspešnost projekta. Vhode in izhode posamezne faze prikazuje slika 5.

*Začetna faza* - projekt definiramo s poslovnega stališča in določimo njegov obseg. V ta namen poiščemo vse zunanje akterje, s katerimi bo sistem sodeloval, ter v grobem definiramo način sodelovanja. Poiskati moramo vse primere uporabe, opišemo pa le najpomembnejše. Poslovno definiranje projekta zajema določitev kriterija uspeha oziroma neuspeha projekta, ocenitev tveganj, presojo potrebnih virov in fazni načrt, iz katerega so razvidni glavni časovni mejniki.

*Faza zbiranja informacij* - analiziramo problemsko področje, postavimo osnovno ogrodje arhitekture sistema, izdelamo projektni plan in razrešimo najbolj rizične elemente projekta. Arhitekturne odločitve morajo upoštevati sistem kot celoto, kar zahteva podroben opis večine primerov uporabe in upoštevanje nekaterih dodatnih omejitev. Prototipno realiziramo sistem do te mere, da lahko prikažemo glavne primere uporabe in ovrednotimo izbrano arhitekturo. Na koncu te faze pregledamo podrobne cilje sistema in nje-

1 *Generacijo programske opreme definiramo kot novo verzijo, ki gre skozi vse faze razvoja.*

### Statični vidik

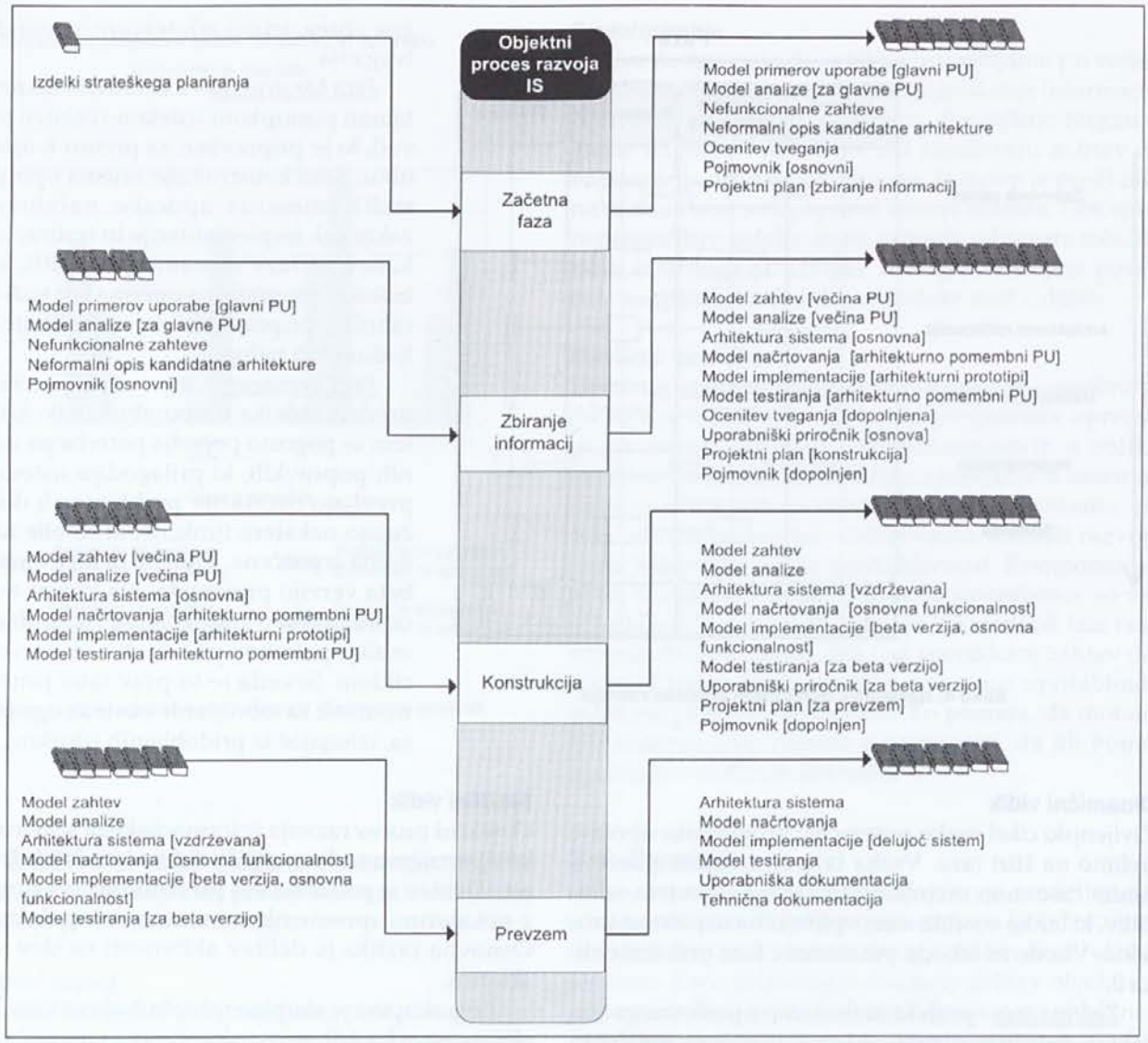
Objektni proces razvoja informacijskega sistema lahko opazujemo tudi s stališča aktivnosti, ki jih izvajamo. Delitev je podobna kot pri strukturnem pristopu z nekaterimi spremembami oziroma dopolnitvami. Osnovna razlika je delitev aktivnosti na dve veliki skupini.

Prva skupina je skupina tehničnih aktivnosti. To so aktivnosti, v katerih se osredotočimo na opravila, ki so popolnoma tehnične narave (npr. identifikacija primerov uporabe, določanje razredov, dodeljevanje odgovornosti).

Druga skupina aktivnosti pa so podporne aktivnosti, ki so namenjene vodenju tehniških aktivnosti in nadzoru projekta. Teh aktivnosti v dokumentu ne bomo opisovali, saj so skupne tako strukturnemu kot objektnemu razvoju. Gre za aktivnosti upravljanja konfiguracij in upravljanja sprememb, aktivnosti projektnega vodenja in aktivnosti upravljanja okolja.

*Zajemanje zahtev* - poglavitni namen je pridobiti uporabnikove zahteve za nastajajoči sistem v obliki primerov uporabe, lastnosti in ne-funkcionalnih zahtev. Izhod te aktivnosti je model primerov uporabe, scenariji primerov uporabe in opis nefunkcionalnih zahtev.

*Analiza* – definira, katere operacije in objekti bodo prisotni v razvijajočem se sistemu, ne oziraje se na to, kako bodo te operacije in objekti izdelani. Izhod iz te aktivnosti zajema razredni diagram in vmesnik sistema,




**Slika 5: Vhodi in izhodi posameznih faz**

ki se odraža v operacijah in dogodkih. Razredni diagram analize oblikujemo na podlagi razrednega diagrama problemskega področja.

*Arhitekturno načrtovanje* – določa strukturo sistema glede komponent in povezav med njimi. Primarni rezultat arhitekturnega načrtovanja je fizična arhitektura sistema, ki izhaja iz logične arhitekture. Logična arhitektura je prvi približek zgradbe sistema, katero dopolnjujemo in spreminjamo z namenom izdelati optimalno arhitekturno zgradbo sistema.

*Načrtovanje objektov* - operacije priredimo objektom in sprejmemo odločitve glede dedovanja, vidljivosti in predstavitve povezav (asociacij). Opišemo tudi pristope k sočasnemu izvajanju posameznih delov sistema in s tem v povezavi tudi sinhronizacijo. Izhodi so razredni diagram načrtovanja, diagram sodelovanja objektov, začetna konfiguracija objektov in posodobljen arhitekturni diagram.

*Implementacija* – rezultate načrtovanja implementiramo v enem izmed programskih jezikov. Pri tem upoštevamo lastnosti izbranega programskega jezika. Potrebno je razviti tako tehnološke kot tudi poslovne razrede. Pri razvoju je potrebno upoštevati izbrano arhitekturo sistema. Upoštevati je potrebno tudi nefunkcionalne zahteve.

*Testiranje* - izvedemo testiranje izdelka. V sklopu te aktivnosti je potrebno izvesti tako testiranje enot kot tudi integracijsko testiranje. Izdelek aktivnosti je poročilo o testiranju, nastane pa tudi množica testnih primerov in testnih vzorcev, s katerimi zagotovimo ponovljivost testiranja. V primeru, ko želimo testiranje avtomatizirati, razvijemo testne razrede.

## 4. Zaključek

Jezik UML prispeva k lažji komunikaciji med razvijalci programske opreme. Uporaba diagramskih tehnik, ki jih jezik UML vsebuje, poenostavlja razumevanje vse bolj zapletenih konceptov gradnje programske opreme, omogoča dokumentiranje dela in izdelkov, gradnjo knjižnic komponent in nenazadnje tudi knjižnic idejnih rešitev skozi vzorce načrtovanja.

Poenotenje izgleda notacijskih konstruktov in natančna določila za njihovo povezovanje so le začetni koraki k boljši programski opremi. Pomembno je, da določimo opravila, ki jim sledimo pri razvoju programske opreme. Opravila povežemo v skupek pravil, strategij, aktivnosti, metod in korakov, katerih namen je doseči poslovne cilje, in jih imenujemo procesni model. Univerzalnega procesnega modela ni, odvisen je od velikosti in tipa organizacije. V prispevku smo opisali ogrodje objektnega procesnega modela, ki lahko služi kot osnova za oblikovanje procesnega modela organizacije.

## 5. Literatura


1. Derr Kurt W., *Applying OMT : A practical step-by-step guide to using the object modeling technique*, SIGS Books, 1995
2. Fowler M, Kendall S, *UML Distilled – Applying the standard object modeling language*, Addison-Wesley, 1997
3. Grady Booch, James Rumbaugh, *Unified Method*, Rational Software Corporation, 1995
4. Inštitut za informatiko, *Enotna metodologija razvoja informacijskih sistemov - 4. Zvezek: Objektni razvoj IS tehnično poročilo*, FERI Maribor Institut za informatiko, november 1999
5. *OMG Unified Modeling Language Specification*, version 1.3, June 1999, http://www.omg.org/
6. Rozman et al., *PROCESSUS - Integration of SEI CMM and ISO quality models*, Software Quality Journal, Marec 1997, str. 37-63
7. Živkovič Aleš, Rozman Ivan, *Značilnosti uspešnih projektov*, OTS'99 zbornik prispevkov, str. 162 - 168


◆


*Dr. Ivan Rozman je redni profesor Univerze v Mariboru, dekan Fakultete za elektrotehniko, računalništvo in informatiko v Mariboru in ustanovitelj Laboratorija za informacijske sisteme, ki ga vodi še danes. Je avtor številnih publikacij in vodja več raziskovalnih projektov. Diplomiral je na Fakulteti za elektrotehniko v Ljubljani, magistriral in doktoriral pa na Tehniški fakulteti v Mariboru.*
*e-pošta: i.rozman@uni-mb.si*

◆

*Dr. Marjan Heričko je docent na Inštitutu za informatiko na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru. Njegovo raziskovalno-razvojno delo obsega vse vidike objektne tehnologije, s poudarkom na metodologijah razvoja, orodjih CASE, razvojnih okoljih in metrikah. Svoja spoznanja in izkušnje je predstavil v številnih prispevkih na domačih in tujih konferencah ter revijah. Aktivno sodeluje pri koordiniranju aktivnosti Centra za objektno tehnologijo ter vodi organizacijo strokovnih srečanj OTS Objektna tehnologija v Sloveniji. Diplomiral, magistriral in doktoriral je na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru.*
*http://lisa.uni-mb.si/osebje/hericko; e-pošta: marjan.hericko@uni-mb.si*

◆

*Mag. Aleš Živkovič je asistent na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru in aktiven član Centra za objektno tehnologijo, v okviru katerega je sodeloval pri pripravi in izvedbi številnih seminarjev in delavnic iz področja Jave, objektnega načrtovanja, porazdeljenih objektnih sistemov in interneta. Je avtor številnih domačih in tujih člankov. Njegovo raziskovalno delo zajema javansko tehnologijo, področja objektne tehnologije, projektnega vodenja in interneta. Je ustanovitelj in koordinator skupine JUGSI (Java User Group of Slovenia). Diplomiral in magistriral je na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru.*
*e-pošta: ales.zivkovic@uni-mb.si*