Centar Informacijske Sigurnosti

Dokumentiranje programskih rješenja

lipanj 2011.

CIS-DOC-2011-06-015

Upozorenje

Podaci, informacije, tvrdnje i stavovi navedeni u ovom dokumentu nastali su dobrom namjerom i dobrom voljom te profesionalnim radom CIS-ovih stručnjaka, a temelje se na njihovom znanju i petnaestak godina iskustva u radu u informacijskoj sigurnosti. Namjera je da budu točni, precizni, aktualni, potpuni i nepristrani.

Ipak, oni su dani samo kao izvor informacija i CIS ne snosi nikakvu izravnu ili posrednu odgovornost za bilo kakve posljedice nastale korištenjem podataka iz ovog dokumenta.

Ukoliko primijetite bilo kakve netočnosti, krive podatke ili pogreške u ovom dokumentu, ili imate potrebu komentirati sadržaj molimo Vas da to javite elektroničkom poštom na adresu info@CIS.hr.

O CIS-u

CIS izrađuje pregledne dokumente (eng. white paper) na teme iz područja informacijske sigurnosti koji će biti korisni zainteresiranoj javnosti, a u svrhu **podizanje njezine svijesti o informacijskoj sigurnosti** i **sposobnosti za čuvanje i zaštitu informacija i informacijskih sustava**. Pored toga, CIS razvija i održava mrežni portal www.CIS.hr kao referalnu točku za informacijsku sigurnost za cjelokupnu javnost; izrađuje obrazovne materijale namijenjene javnosti; organizira događaje za podizanje svijesti o informacijskoj sigurnosti u javnosti i pojedinim skupinama te djeluje u suradnji sa svim medijima.

CIS **okuplja mlade** zainteresirane za informacijsku sigurnost i radi na njihovom pravilnom odgoju i obrazovanju u području informacijske sigurnosti te pripremu za **profesionalno bavljenje informacijskom sigurnošću**.

Centar informacijske sigurnosti [CIS] nastao je 2010. godine na poticaj Laboratorija za sustave i signale[LSS] Zavoda za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu, a kao posljedica 15togodišnjeg rada na istraživanju, razvoju i primjeni informacijske sigurnosti. LSS je među ostalim potaknuo osnivanje CARNetovog CERTa i sudjelovao u izradi Nacionalnog programa informacijske sigurnosti RH.

Smisao CISa je da bude **referentno mjesto za informacijsku sigurnost** za javnost, informatičare i posebno za mlade te da sustavno podiže njihovu svijest i sposobnosti u području informacijske sigurnosti.

Rad CISa podržava Ministarstvo znanosti, obrazovanja i sporta Republike Hrvatske, a omogućuju sponzori.

Prava korištenja

Sadržaj

1. Uvod

Svi veliki programski projekti, bez obzira na svrhu za koju se razvijaju, stvaraju velike količine dokumentacije. Za projekte srednje veličine, količina tiskane dokumentacije će zasigurno popuniti nekoliko polica, dok će za one velike projekte ta ista dokumentacija ponekad ispuniti i nekoliko soba. Tijekom raznih faza razvoja projekta, izrada dokumentacije je financijski i vremenski zahtjevan zadatak, te zbog toga predstavlja popriličnu komponentu u budžetu koji je predviđen za sam projekt.

Postoji niz prepreka u izradi kvalitetne dokumentacije, a jedna od njih je relativno kratko vrijeme koje se daje za njezinu izradu. U većini današnjih projekata zahtjevi se brzo mijenjaju i potrebno je brzo reagirati kako projekt ne bi ispaštao u kvaliteti. Ovakvo dinamičko okruženje često uzrokuje zanemarivanje dokumentacije. Ovim dokumentom se želi ukazati na važnost programske dokumentacije, a daje se i pregled tipova dokumentacije te preporuke za njenu kvalitetnu izradu.

Dokumentacija programskog projekta ima višestruke uloge. Na primjer, može služiti kao komunikacijski medij između članova razvojnog tima ili kao priručnik koji će korisnici moći koristiti prilikom rada i administracije sustava. Također, može poslužiti kao jedinstveni repozitorij informacija razvojnim inženjerima koji održavaju sustav. Često se koristi kao izvor informacija menadžmentu koji će na temelju tih informacija donositi odluke vezane uz budžet, poslovni plan i daljnju razradu rasporeda razvoja projekta.

Za zadovoljavanje svih navedenih potreba obično nije dovoljno napisati jedan već više vrsta dokumenata. Često se počinje od neformalno napisane radne dokumentacije pa sve do profesionalno obrađenih korisničkih uputa. Ovisno o namijeni postoji nekoliko tipova dokumentacija, a u ovom dokumentu se opisuju najvažnije. Dokumentacije u ovom dokumentu su odabrane prema vodećim referencama iz područja programskog inženjerstva (u dodatnoj literaturi pod [1] – [8]).

2. Programska dokumentacija

U ranim fazama razvoja računarstva programska potpora je bila vezana za računalo na kojemu je napravljeno. Tada nije bilo potrebno voditi računa o prenošenju programa na druge odjele organizacije ili na veće udaljenosti jer su računala bila slabo rasprostranjena. Razvojem modernog računarstva nastale su platforme pomoću kojih se programska potpora mogla prenositi s jednog računala na drugo. Ovakav preokret pokrenuo je naglu informatizaciju i integraciju računala u gotovo sve grane današnjeg modernog društva.

Pokretni programski kod donio je sa sobom određene prepreke. Naime, trebalo je prenijeti određen dio znanja zajedno s programskom potporom kako bi ga drugi korisnici (ili razvijatelji) mogli preuzeti i koristiti. Dokumentacija programske potpore u ranim fazama razvoja računarstva se uglavnom zasnivala na korisničkim uputama. Druga faza razvoja dokumentacije nastala je zajedno sa zajednicom otvorenog koda. Iako je u početcima bila vrlo složena predstavljala je prvi korak u stvaranju jedinstvenog jezika koji je omogućavao stručnjacima diljem svijeta da opisuju svoja programska rješenja. Složena programska dokumentacija je bila namijenjena isključivo iskusnim stručnjacima koji su iz dokumentacije mogli shvatiti tek osnovne detalje programa.

Konačna faza razvoja je nastala korištenjem službenih dokumentacija u komercijalnim programskim proizvodima. Velike komercijalne organizacije su ubrzo shvatile da je potrebno dokumentirati svoje proizvode iako većina dokumentacije nije namijenjena javnosti. Točnije, izrada programske dokumentacije u komercijalnim projektima postala je važan dio programskog inženjerstva jer su programi postajali sve veći i složeniji. Velika i složena programska rješenja postala su zahtjevna za održavanje. Dodatno, složena programska rješenja postala su ovisna o programerima koji su ih razvili jer je obuka novih programera predstavljala velik trošak. Naime, osim znanja o svim korištenim tehnologijama i alatima, novi programeri morali su razumjeti način razmišljanja programera koji je razvio program. Bez kvalitetne programske dokumentacije novi programeri nisu mogli održavati postojeći projekt i nadograđivati ga. Mnoge organizacije su uočile ovaj problem i razvile svoje metode stvaranja dokumentacije. Ovisno o tipu organizacije i programskoj potpori koja se razvijala nastali su razni oblici dokumentacija. Najbitnije dokumentacije i način njihove izrade opisane su u nastavku poglavlja, a više informacija može se pronaći u dodatnoj literaturi pod [1], [2], [3], [5] i [6].

2.1. Dokumentacija zahtjeva

Dokumentiranje korisničkih zahtjeva je jedno od osnovnih koraka u razvoju programske potpore. Gotovo sve moderne metode razvoja programske potpore (npr. *RUP*, *PRINC*) početne korake zasnivaju na na izradi detaljne dokumentacije svih korisničkih zahtjeva. Naravno, korisnički zahtjevi su podložni promjenama. Iz tog razloga zahtjeva se dokumentaciju koristi kao medij za komunikaciju s korisnikom kako bi se utvrdili konkretni zahtjevi na sustav. U nekim slučajevima zahtjevi nisu poznati za vrijeme izrade sustava. Na primjer, kada korisnik nema dovoljno dobar uvid u problem koji želi riješiti programskom potporom ili se pak ne razumije dovoljno dobro u svoju domenu. U takvim situacijama zahtjevi se prikupljaju opsežnim intervjuiranjem korisnika (i osoblja ukoliko postoji) te izradom konkretnih scenarija primjene koje korisniku pomažu u predočavanju konkretnih zahtjeva. Jednom prikupljeni zahtjevi se bilježe u posebnom dokumentu koji se predaje razvojnom timu zaduženom za implementaciju zahtjeva.

Dokumentacija zahtjeva opisuje zašto je program potreban i stavlja ga u kontekst koji opisuje kako može pomoći korisniku te kako i što će konačan proizvod raditi. Velik dio dokumentacije zahtjeva se odnosi na popis konkretnih korisničkih zahtjeva (opisani u nastavku). Zahtjevi uključuju svojstva sustava, opis rada sustava pa čak i ograničenja na proces izrade. U općem slučaju, zahtjevi su izjave o tome što i kako bi sustav trebao raditi. Zahtjevi koji opisuju kako sustav treba raditi se odnose na dizajn sustava i trebalo bi ih odvojiti u zaseban dokument (ili barem poglavlje) jer ih je potrebno odvojiti od korisničkih zahtjeva radi kvalitetnije komunikacije s korisnikom. Naime, korisnici često ne žele znati tehničke detalje u oblikovanju sustava već samo krajnji rezultat. Miješanje sustavskih i korisničkih zahtjeva može otežati komunikaciju s korisnikom te usporiti početnu fazu prikupljanja i analiziranja zahtjeva. Ovisno o tome tko su krajnji korisnici, konkretni zahtjevi se mogu međusobno razlikovati. Krajnji korisnici svoje zahtjeve obično iznose opisnim rečenicama. Na primjer, korisnici bi tražili određenu funkcionalnost na sljedeći način: „program mora omogućiti unos informacija o novim proizvodima i automatski

ažurirati stanje na skladištu, a sve to se treba birati putem odgovarajućih izbornika“. Takvi zahtjevi se često moraju razgraditi u više odvojenih zahtjeva kako bi se postigla kvalitetna raspodjela posla. Točnije, primjer obuhvaća zahtjev za grafičkim sučeljem, bazom podataka i određenim ponašanjem sustava.

Drugi oblik zahtjeva dolazi od klijenata koji nisu krajnji korisnici već naručitelji programske potpore. Njihovi zahtjevi se često odnose na vremenske rokove i trošak razvoja sustava. Dodatno, zahtjevi se mogu odnositi i na performance sustava. Na primjer, sustav na zadanom poslužitelju mora moći obraditi najmanje 1.000 korisničkih zahtjeva u sekundi. Bitno je sve ove različite tipove zahtjeva uzeti u obzir prilikom stvaranja dokumentacije zahtjeva kako bi razvojni tim mogao kvalitetno odabrati platformu i arhitekturu.

Dodatna prepreka u kvalitetnom dokumentiranju zahtjeva je određivanje važnosti ili prioriteta. U složenim sustavima može nastati velik broj zahtjeva koji se međusobno isprepliću ili su pak potpuno odvojeni. Ukoliko postoje konflikti među zahtjevima potrebno im je dodijeliti više pažnje od drugih. Ispravno postavljanje prioriteta je važan korak u izradi dokumentacije zahtjeva i zato je potrebno u suradnji s korisnicima (ili klijentima) odrediti važnosti svih zahtjeva.

2.1.1. Korisnički zahtjevi

Korisnički zahtjevi definiraju što konkretno sustav treba raditi i kako će pomoći krajnjim korisnicima prilikom rada. Stvaraju ih krajnji korisnici opisivanjem svojih zaduženja na radnim mjestu. Iz tog razloga se ovakvi zahtjevi često nazivaju i funkcijskim zahtjevima budući da definiraju novu funkcionalnost sustava. Shvaćanje korisničkih zahtjeva je ključan korak u izradi i uspjehu programskih rješenja. Današnje metodologije razvoja programske potpore podržavaju upravljanje i prikupljanje zahtjeva. Tablica u nastavu je primjer kako izgleda popis korisničkih zahtjeva za programsku potporu turističke agencije.

Većina modernih metodologija razvoja proces prikupljanja korisničkih zahtjeva opisuje kroz nekoliko faza. Ovisno o konkretnoj metodologiji pojedina faza može biti drugačija ili pripojena drugoj fazi. No, osnova je gotovo uvijek ista, a u nastavku se opisuju četiri općenite faze (više informacija moguće je pronaći u dodatnoj literaturi pod [7] i [11]).

- **Prikupljanje informacija** – prvi korak prilikom utvrđivanja korisničkih zahtjeva je prikupljanje informacija. U ovoj fazi se radi istraživanje tržišta kako bi se dobio bolji uvid u zahtjeve. Identificiraju se svi dionici[1] na koje će sustav imati utjecaj te njihove konkretne uloge u konačnom sustavu. Potrebno je i identificirati kontekst uporabe konačnog sustava. Točnije, koja svojstva bi trebao imati, kada i u kojim slučajevima će se koristiti te koja svojstva su ključna za kvalitetniji rad.
- **Utvrđivanje korisničkih potreba** – u ovoj fazi se provodi intervjuiranje svih dionika koji su uključeni u projekt. Prilikom intervjuiranja često se korisnike provodi kroz scenarije kojima se opisuje hipotetska situacija za koju korisnik izriče kako želi da se programska potpora ponaša. Ovisno o složenosti projekta i količini zahtjeva mogu se organizirati grupni intervjui. Ova metoda utvrđivanja zahtjeva je korisna jer se korisnici u grupi osjećaju slobodnije i mogu poticati jedan drugog u izricanju ideja.
- **Planiranje i vrednovanje** – kada se prikupi dovoljno informacija moguće je početi s procesom utvrđivanja korisničkih potreba. To može biti dugotrajna faza budući da se treba proći kroz sve prikupljene informacije i razumjeti što je korisnik htio reći. Kada se izluči dovoljno zahtjeva potrebno je izgraditi prototip kojim će se provjeriti da li odgovara stvarnim potrebama korisnika
- **Specifikacija zahtjeva** – kada se jednom svi korisnički zahtjevi definiraju i utvrde potrebno ih je dokumentirati na odgovarajući način kako bi se mogli održavati. Dodatno, zahtjeve je potrebno kategorizirati po domeni kojoj pripadaju (npr. zahtjevi sučelja, funkcionalnosti) te po važnosti ili prioritetu

[1] Dionik (engl. *Stakeholder*) – osoba koja može osjetiti utjecaj posljedica određenih akcija u razvoju projekta. Općenito se odnosi na osobe koje ulažu u projekt ili ga naručuju, te imaju određenu korist od njega.

Zahtjev ID	Opis zahtjeva
KZ 1.1	Korisnik mora imati mogućnost odabira jednog ili više turističkih odredišta
KZ 1.2	Prilikom odabira, korisnik može odabrati kojim prijevoznim sredstvom želi putovati (autobus, vlak, zrakoplov, brod)
KZ 1.3	Nakon odabira odredišta, korisniku treba napraviti ponudu koja se može isprintati na papir A4 formata

2.1.2. Zahtjevi na sustav

Sustavski zahtjevi se razlikuju od korisničkih zahtjeva po tome što ne definiraju novu funkcionalnost. Iz tog razloga se ponekad nazivaju i nefunkcionalni zahtjevi jer se njima uglavnom izriču ograničenja ili uvjeti rada samog sustava. Sustavskim zahtjevima se opisuju tehnička svojstva sustava ali, za razliku od tehničke dokumentacije, u specifikaciji sustavskih zahtjeva se samo bilježi potreba za pojedinim obilježjima. U tehničkoj dokumentaciji se zatim opisuje kako je pojedinih zahtjev ostvaren. Tablica u nastavku daje se primjer sustavskih zahtjeva za programsku potporu turističke agencije. Neke skupine sustavskih zahtjeva dane su u nastavku, a više informacija se može pronaći u dodatnoj literaturi pod [7] i [10].

- **Sklopovski zahtjevi** – opisuju sklopovsku okolinu na kojoj programska potpora treba raditi. Sklopovski zahtjevi često određuju konkretnu platformu koja će se koristiti u izradi programske potpore. Na primjer, ukoliko programska potpora mora raditi brzo na slabim računalima potrebno je koristiti jezike niže razine kao C/C++. Također, ako sustav treba raditi sa što manjom opskrbom energije i slabim radnim taktom izabrat će se jezik najniže razine, odnosno asembler.
- **Kvaliteta usluge** – ovim zahtjevima se opisuje koliko dobro će sustav raditi u iznimnim situacijama. Na primjer, koliko će se korisnika obraditi u jedinici vremena. Kvaliteta usluge je često vezana i uz sklopovske zahtjeve i zato je klijentima potrebno istaknuti sve moguće konflikte između te dvije skupine.
- **Ovisnost o vanjskim sustavima** – ukoliko sustav koristi javno sučelje drugog sustava potrebno je ovu ovisnost istaknuti u dokumentaciji. Takve ovisnosti često mogu biti problematične budući da se stvara ovisnost o sustavu koji nije dio vlastitog projekta. Ukoliko je moguće, potrebno je u dokumentaciji istaknuti sve moguće alternative za zadani sustav kako bi se po potrebi on mogao zamijeniti.

Zahtjev ID	Opis zahtjeva
SZ 1.1	Programska potpora se mora moći pokrenuti na Pentium 4 ili boljim procesorima
SZ 1.2	Sustav mora imati javno sučelje koje će drugim sustavima omogućiti pozivanje pojedinih funkcionalnosti i dobivanje informacija
SZ 1.3	Glavni poslužitelj mora obavljati barem 400 korisničkih zahtjeva u sekundi
SZ 1.4	Sustav treba biti skalabilan za slučaj da će se dodavati novi poslužitelji

2.2. Tehnička dokumentacija

Tehnička dokumentacija predstavlja jedinstveni jezik komunikacije između razvijatelja programske potpore i ostalih stručnjaka koji imaju interes za projektom. Iz tog razloga ona čini jedan od najvažnijih oblika dokumentacije programske potpore. Ukoliko tehnička dokumentacija nije dostupna drugi programeri i razvijatelji moraju sve informacije pronalaziti iz izvornog koda programske potpore.

Dodatna prepreka u izradi tehničke dokumentacije je održavanje konzistentnosti između dokumentacije i programske potpore. Naime, tehnička dokumentacija se često održava zasebno, izvan programskog koda (najčešće u obliku tekstualnog dokumenta). Korisnički zahtjevi su podložni čestim promjenama, a samim time se i programski kod mora mijenjati i nadograđivati kako bi zadovoljio sve potrebe korisnika. Ovakve česte izmjene mogu u potpunosti izmijeniti ponašanje programa i unutarnju organizaciju. Ukoliko se sve promjene u projektu ne zabilježe kroz tehničku dokumentaciju ona prestaje biti ažurna. Takva dokumentacija je vrlo štetna za projekt jer daje krive informacije o trenutnom stanju programske potpore. Kako bi se ovo izbjeglo koriste se razne metode proizvođenja programske dokumentacije koja osigurava ažurnost dokumentacije s programskim modelom. Neke od tih metoda se analiziraju u poglavlju 3.2., a više informacija moguće je pronaći u dodatnoj literaturi pod [2] i [3].

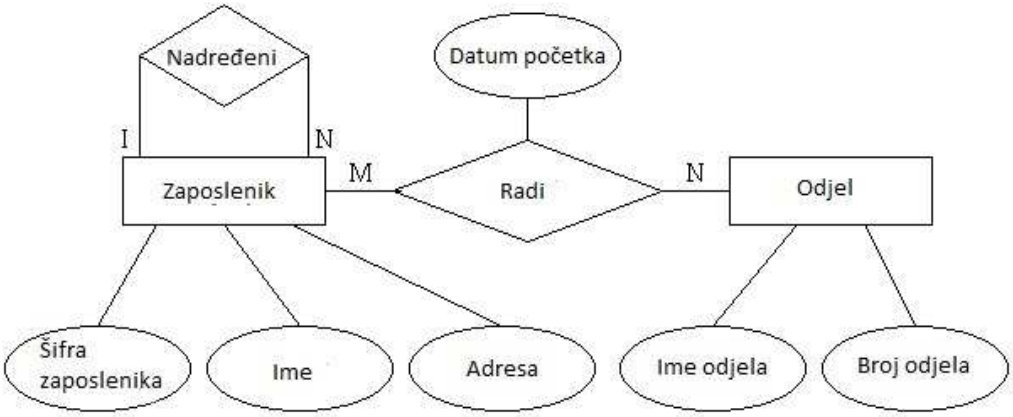
2.2.1. Dokumentacija baze podataka

Baze podataka čine važan dio mnogih sustava, zato ih je potrebno kvalitetno dokumentirati kako bi se mogle održavati za vrijeme životnog ciklusa projekta. Dokumentacija baze podataka se ističe dijagramima koji opisuju pojedine tablice i njihove ovisnosti te opis pojedinih atributa. Često se koriste *ER* (engl. *Entity Relationship*) dijagrami (Slika 1.) i dijagrami tablica kako bi se istaknule veze između tablica. Najzastupljenije baze podataka su relacijske baze pa se iz tog razloga razmatraju u nastavku poglavlja.

Prilikom dokumentacije pojedinih tablica bitno je istaknuti značenje pojedinih atributa i njihovih vrijednosti. Na primjer, ukoliko neki atribut nije postavljen (ima vrijednost *NULL*) to ima različito značenje od vrijednosti nula. Budući da se konkretno značenje vrijednosti atributa ne može uočiti iz konteksta tablice potrebno je dokumentirati sva posebna značenja ukoliko ih ima. Postoje različiti načini dokumentiranja tablica, a jedna od najčešće korištenih metoda je popisivanje pojedinih obilježja prema jasno definiranom predlošku. Konzistentno korištenje predloška za dokumentiranje pojedinih tablica je važno budući da se time olakšava snalaženje u dokumentaciji. Ovo posebice dolazi do izražaja kada baza podataka ima velik broj tablica. Primjer predloška se nalati u nastavu u tabličnom obliku.

Kako je razvoj i održavanje baze podataka složen zadatak većina razvojnih timova imaju nekoliko članova koji se brinu o njoj. Na primjer, članovi tima mogu imati uloge:

- **Arhitekt baze podataka** – zadaća arhitekta je izrada teorijskog modela baze podataka, što uključuje definiranje tablica i ovisnosti među njima, očuvanje integriteta baze te sprječavanje pojavljivanja redundancije (višestrukosti podataka).
- **Administrator** – održava bazu podataka te unosi izmjene u bazi ako su potrebne. Njegova uloga je jako važna jer životni vijek baze podataka uvelike ovisi o kvaliteti njenog održavanja, što je posao administratora.



Slika 1. Primjer ER modela

Izvor: http://www-inf.int-edu.eu/cours/WebServices/XML/SBD-Eng/tpasrea.html

Naziv tablice	app_korisnici	
Svrha tablice	sadrži popis korisnika koji imaju pravo prijave na sustav	
Primarni ključ	korisnik_id	
Strani ključ/evi	-	
Naziv atributa	**Opis atributa**	**Tip podataka**
Naziv	Korisničko ime korisnika	TEXT
Lozinka	Lozinka šifrirana MD5 sažetkom	TEXT
Admin	Označava da li je korisnik administrator. Ukoliko je bilo koja vrijednost različita od nula ili NULL, korisnik je administrator	TINYINT

2.2.2. Dokumentacija izvornog koda

Danas se za dokumentiranje programskog koda najčešće koriste automatizirani alati. Neki od najkorištenijih alata za izradu komentara su opisani u poglavlju 4.2. Ti alati na temelju komentara stvaraju tehničku dokumentaciju u HTML (engl. *HyperText Markup Language*), PDF (engl. *Portable Document Format*) i drugim oblicima. Tako organizirani dokumenti omogućuju programeru da brzo pronađe definiciju neke funkcije ili razreda. Iako se kvalitetnom dokumentacijom želi olakšati razumijevanje sustava, ponekad je potrebno proučavati konkretni programski kod. Izvorni kod je također bitan dio dokumentacije budući da se stvarno ponašanje sustava krije upravo u njemu. Iz tog razloga se velik dio vremena provodi na refaktoriranje programskog koda. Refaktoriranje je proces restrukturiranja programskog koda u kojemu se ne dodaju nove funkcionalnosti već se preoblikuje kod kako bi bio lakši za razumijevanje i održavanje.

Dobra i čitljiva struktura programskog koda se ponekad cijeni više od programske dokumentacije. Na primjer, organizacija *Google* ne piše komentare u programskom kodu jer zauzimaju puno prostora u velikim projektima i teško ih je održavati. Ukoliko se dokumentacija napravi prije konkretnog koda teško je predvidjeti što će se točno raditi u pojedinim klasama i funkcijama. Ako se prvo napravi implementacija pa tek onda krene u izradu dokumentacije postoji rizik da određeni dijelovi nikada neće biti dokumentirani ili da neće biti ažurni. Kako se sve informacije nalate u samom kodu, on je dovoljan za razumijevanje strukture i toka podataka u programu.

Postoji nekoliko praksi koje je dobro koristiti prilikom strukturiranja programskog koda. Neki od tih praksi su u nastavku, a više informacija o dobrim praksama i metodama pisanja izvornog koda može se pronaći u dodatnoj literaturi pod [5].

- **Školovano programiranje** (engl. *Literate Programming*) – metoda koju je razvio Donald Knuth, računalni znanstvenik, kao alternativu strukturiranom programiranju. Ovom metodom se programski kod oblikuje tako da bude sličan ljudskom načinu razmišljanja, a ne računalu. Naime, u metodi školovanog programiranja kod se oblikuje kao niz logičkih izjava u ljudskom jeziku (a ne u programskom jeziku). Iako ova metoda olakšava razumijevanje koda ne koristi se u današnjim programskim jezicima. Više informacija o ovoj metodi moguće je pronaći u dodatnoj literaturi pod [13].
- **Strukturirano programiranje** (engl. *Structured Programming*) – predstavlja uobičajenu programsku paradigmu koja se koristi u svim popularnim programskim jezicima. Najpoznatija je po izbjegavanju *GOTO* naredbe koja je uzrokovala veliku količinu nepreglednosti u izvornom kodu. Ova metoda se danas najčešće koristi i zahtjeva veliku količinu komentara i dobrih praksi. Dobre prakse se odnose na dobro imenovanje funkcija, razreda i entiteta (kao u poglavlju 3.3), sažimanje koda kako bi bio čitljiviji te odvajanje odsječaka u funkcije koje se mogu ponovno iskoristiti. Više informacija moguće je pronaći u dodatnoj literaturi pod [14].

2.3. Korisnička dokumentacija

Za razliku od tehničke dokumentacije, korisnička dokumentacija je namijenjena krajnjim korisnicima sustava. Korisnička dokumentacija ne sadrži objašnjenja programskog koda niti opis samog dizajna sustava. Njezin cilj je na jednostavan način objasniti kako koristiti pojedine funkcionalnosti programske potpore.

Kako korisnička dokumentacija nije namijenjena informatičkom osoblju važno je izbjegavati tehnički žargon i složene sheme te skice sustava. Ukoliko je programsko rješenje vrlo složeno poželjno je koristiti slike prilikom opisivanja pojedinih koraka. Za ovakve dokumente nije presudno da budu organizirani po nekoj strogoj shemi, ali je ipak važno proizvesti pregledno kazalo te teme objasniti na jednostavan način.

Isto kao i tehnička dokumentacija, bitno je da korisnička dokumentacija bude uvijek ažurna i sinkronizirana s programskom potporom. Ukoliko se promjenom programa (ili zahtjeva) određeni slijed akcija promijeni potrebno je odmah ažurirati korisničku dokumentaciju. Kada korisnička dokumentacija nije ažurna, korisnik neće moći koristiti programsku potporu što će uzrokovati nezadovoljstvo ili čak zastoj u radu. Postoji nekoliko temeljnih oblika korisničke dokumentacije, neki od njih se navode u nastavku, a više detalja može se pronaći u dodatnoj literaturi pod [8].

- **Vodič** – optimalni oblik za novog korisnika, objašnjava sve funkcionalnosti sustava korak po korak na sistematičan i pregledan način. U ovom obliku korisničke dokumentacije pretpostavlja se da korisnik ima tek osnovno znanje o računalima i da se do sada nije susreo s pojedinim dijelovima sustava,
- **Tematska organizacija** – pojedina poglavlja ovakvog dokumenta su organizirana na način da se detaljno objašnjava samo jedno područje interesa. Pretpostavlja se da je korisnik upoznat barem s osnovama rada u sustavu. Na primjer, kada se radi o tematskoj organizaciji dokumentacije programske potpore za uređivanje teksta, jedno poglavlje dokumenta će opisivati kako za ispravno uređeni tekst proizvesti sadržaj. U tom slučaju, pretpostavlja se da korisnik zna ispravno urediti tekst.
- **Lista referenci** – oblik u kojem su sve naredbe ili neki drugi elementi programa detaljno objašnjeni i indeksirani po abecednom redu. Ovaj oblik koriste iskusni korisnici koji su već upoznati s radom programa. Slika 2. prikazuje primjer liste referenci.

Slika 2. Lista referenci

Izvor: http://www.cups.org/doc-1.1/sum.html#CONTENTS

3. Standardi i dobre prakse

Dokumentiranje programskih rješenja je važan dio izrade programske potpore. Kako je izrada programske potpore inženjerska disciplina razvijeni su razni standardi i dobre prakse koje pospješuju pravilno dokumentiranje programskih rješenja. U nastavku poglavlja su navedeni neki standardi i dobre prakse dokumentiranja programske potpore, a više informacija može se pronaći u dodatnoj literaturi pod [5] i [9].

3.1. UML

Kako je opisano u poglavlju 2, dokumentacija je potrebna kako bi se prenijelo određeno znanje od jednog programera do drugog. Kako bi se to omogućilo potrebno je definirati zajednički jezik pomoću kojeg će programeri međusobno komunicirati i dijeliti znanje o proizvedenoj programskoj potpori. Slično kao i sheme tranzistora i strujnih krugova, potrebno je napraviti vizualni jezik za opisivanje programske potpore. Jedan od takvih jezika je *UML* (engl. *Unified Modeling Language*) koji predstavlja jezik za vizualizaciju, specificiranje, konstruiranje i dokumentiranje programske potpore. Neke prednosti korištenja UML standarda su:

- omogućavanje vizualnog modeliranja koje omogućuje izradu i održavanje modela programske potpore u kratkom vremenu,
- pružanje mogućnosti proširivanja i stvaranja specijaliziranih dijelova programske potpore korištenjem oblikovnih obrazaca i njihovom vizualizacijom,
- određena razina nezavisnosti od konkretnog programskog jezika[2],
- podrška razvoju objektno orijentiranih programskih rješenja[3].

UML standard definira više dijagrama, a svaki od njih pomaže u dokumentiranju nekog aspekta programske potpore. Standard definira dvije općenite kategorije dijagrama, a svaka kategorija sadrži konkretne dijagrame od kojih svaki ima drugačija svojstva. Kategorije koje definira *UML* standard su opisani u nastavku.

3.1.1. Statički UML dijagrami

Statički *UML* dijagrami se koriste za opisivanje strukture sustava. Kako i samo ime predlaže, statički dijagrami modeliraju elemente koji su neovisni o vremenu. Točnije, ne prikazuju vremensku ovisnost između pojedinih elemenata u dijagramu (npr. komunikacija dvaju modula, redoslijed poruka u vremenu i drugo). U nastavku su prikazani najkorišteniji statički *UML* dijagrami, a više informacija može se pronaći u dodatnoj literaturi pod [4].

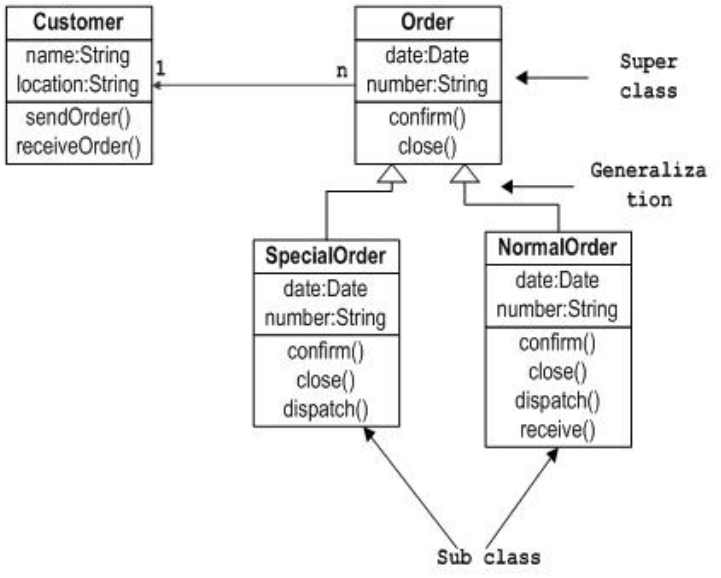
- **Dijagram razreda** (engl. *Class diagram*) – predstavlja jedan od najvažnijih i najkorištenijih *UML* dijagrama općenito. Koristi se isključivo u objektno orijentiranim jezicima, a prikazuje razrede u programu i veze između njih. Dijagram razreda predstavlja jedini *UML* dijagram koji se može izravno preslikati u programski kod putem raznih alata (prikazanih u poglavlju 4). Slika 3. prikazuje dijagram razreda.
- **Dijagram komponenata** (engl. *Component diagram*) – prikazuje međusobnu ovisnost većih gradivnih dijelova ili sustava. Koriste se za opisivanje strukture proizvoljno složenih programskih rješenja. Dijagram komponenata omogućava

[2] Iako je neovisnost o programskom jeziku jedna od često spomenutih prednosti UML standarda, u praksi ovo svojstvo rijetko kada dolazi do izražaja. Naime, za izradu kvalitetne dokumentacije korisnije je napraviti UML dijagrame na temelju izvornog koda aplikacije. Naravno, u fazi planiranja kada arhitektura nije strogo zacrtana UML dijagrami se mogu prilagoditi za bilo koji programski jezik.

[3] Objektno orijentirana programska paradigma predstavlja nov način izrade programske potpore koji se zasniva na objektima, tj. modeliranju stvarnih entiteta iz prirode.

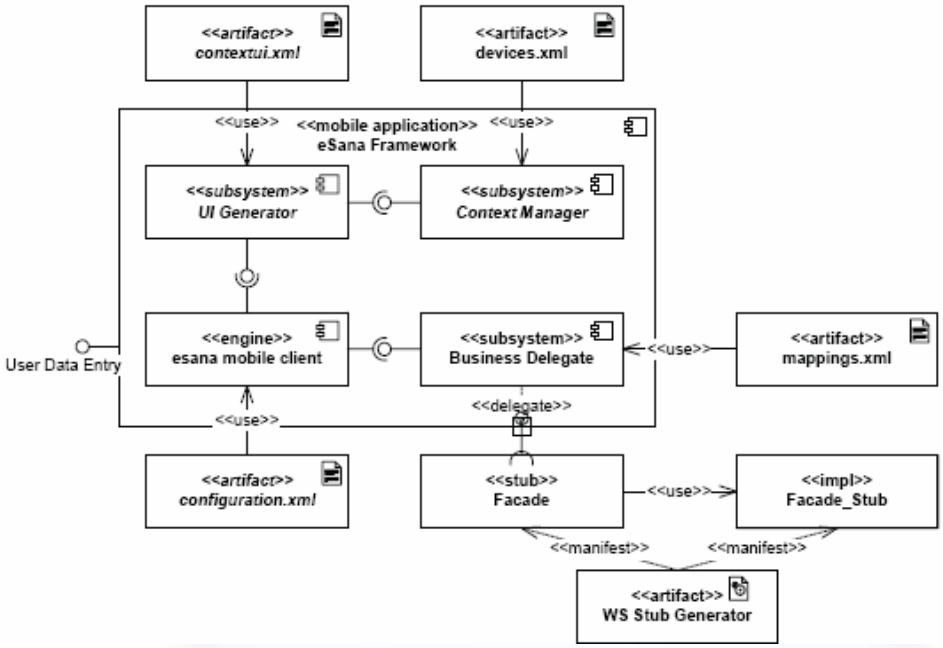
prikaz komponenti s proizvoljnom razinom detalja. Naime, dijagramom se može opisati odnos jednog složenog modula s ostalim složenim modulima, ali moguće je prikazati i odnos unutarnjih komponenti složenog modula. Slika 4. prikazuje dijagram komponenata koji opisuje strukturu i ovisnost jednog složenog modula i više manjih.

- **Dijagram paketa** (engl. *Package diagram*) – opisuje kako je sustav podijeljen u logičke cjeline te prikazuje ovisnosti između njih. Većina programskih jezika koristi određeni sustav za odvajanje logičkih cjelina. Na primjer, programski jezici *C++* i *C#* koriste prostore imena (engl. *Namespace*) za organizaciju logičkih cjelina, dok programski jezik *Java* koristi pakete.
- **Slučajevi korištenja** (engl. *Use case diagram*) – predstavljaju *UML* dijagrame koji opisuju interakciju korisnika s pojedenim dijelovima sustava. Uglavnom se koriste kao sredstvo komunikacije s klijentom prilikom prikupljanja i analize zahtjeva. Konkretni elementi koji se prikazuju nisu nužno stvarne komponente sustava (npr. moduli ili razredi) već apstraktni elementi koji se koriste samo kao opis funkcionalnosti. Slika 6. prikazuje dijagram slučajeva korištenja.



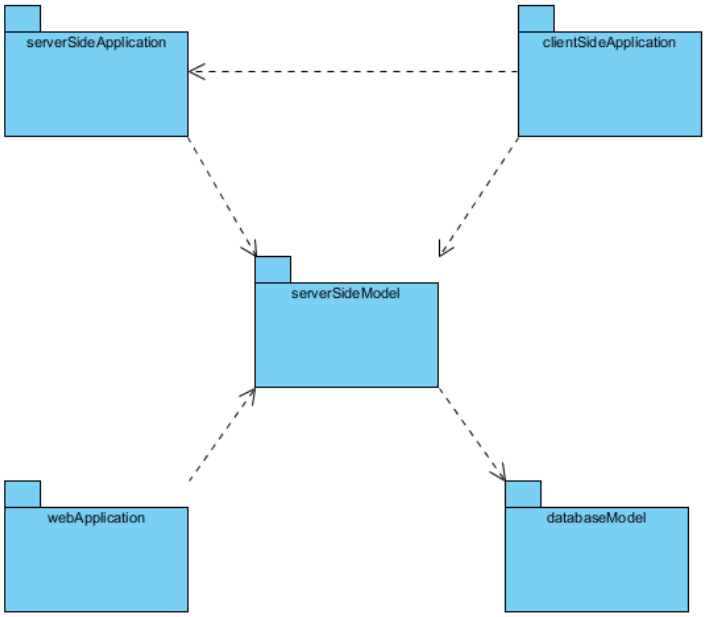
Slika 3. Dijagram razreda

Izvor: http://www.tutorialspoint.com/uml/uml_class_diagram.htm



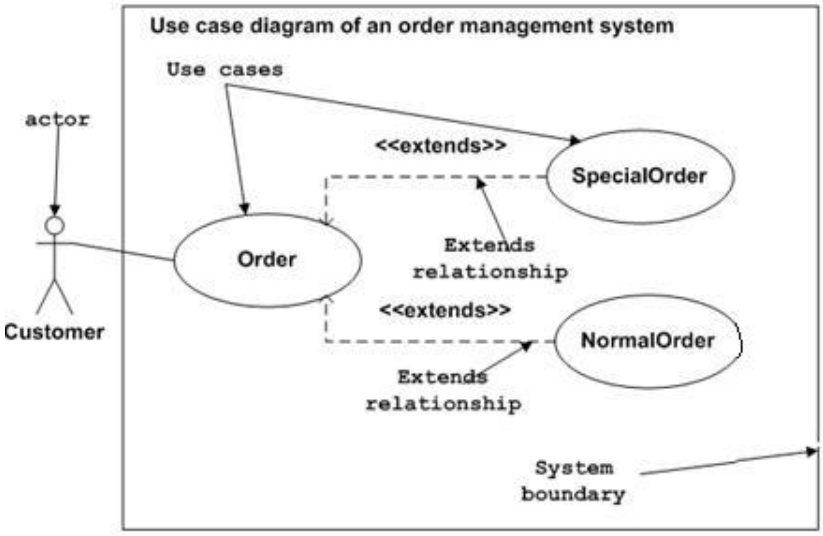
Slika 4. Dijagram komponenata

Izvor: http://diuf.unifr.ch/main/is/esana-framework



Slika 5. Dijagram paketa

Izvor: http://www.visual-paradigm.com/VPGallery/diagrams/Package.html



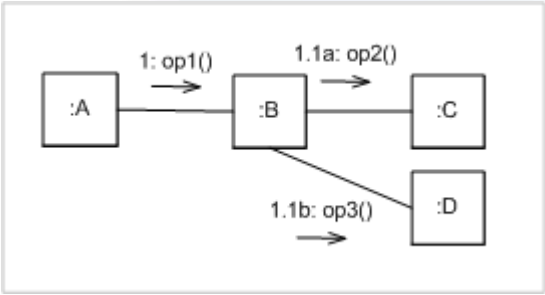
Slika 6. Dijagram slučaja korištenja

Izvor: http://www.tutorialspoint.com/uml/uml_use_case_diagram.htm

3.1.2. Dinamički UML dijagrami

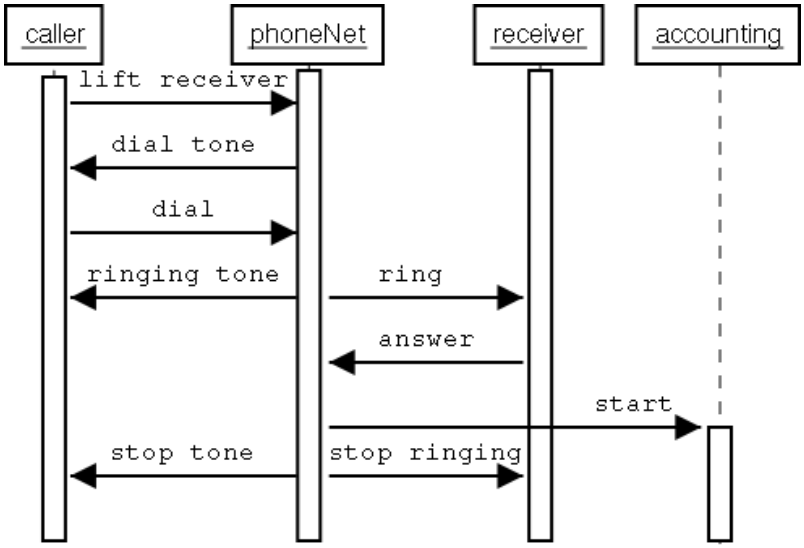
Dinamički *UML* dijagrami prikazuju ovisnost pojedinih elemenata sustava u vremenu te njihovu interakciju. Koriste se kako bi se prikazale komunikacijske veze između pojedinih modula ili za prikaz interakcije s drugim sustavima. Neki od poznatijih dinamičkih *UML* dijagrama su:

- **Dijagram komunikacije** (engl. *Communication diagram*) – prikazuje interakciju između objekata ili njegovih dijelova u obliku razmjene poruka. Poruke su numerirane kako bi se prikazao redoslijed slanja i primanja pojedinih poruka. Dodatno, poruke sadrže pozivajuće funkcije i operande ukoliko ih ima. Slika 7. prikazuje dijagram komunikacije.
- **Dijagram slijeda** (engl. Sequence diagram) – koristi se za prikaz toka izvođenja pojedine akcije u sustavu. Za razliku od dijagrama komunikacije, prikazuje kada se pojedini objekti stvaraju i kada se uništavaju. Koristi strogu strukturu za prikaz pojedinih entiteta koji komuniciraju. Slika 8. prikazuje strogu strukturu dijagram slijeda.



Slika 7. Dijagram komunikacije

Izvor: http://en.wikipedia.org/wiki/Communication_diagram



Slika 8. Dijagram slijeda

Izvor: ***http://www.pms.ifi.lmu.de/publikationen/diplomarbeiten/Sacha.Berger/THESIS/HTML-view/part2.html***

3.2. Korištenje razvojnog okruženja u izradi dokumentacije

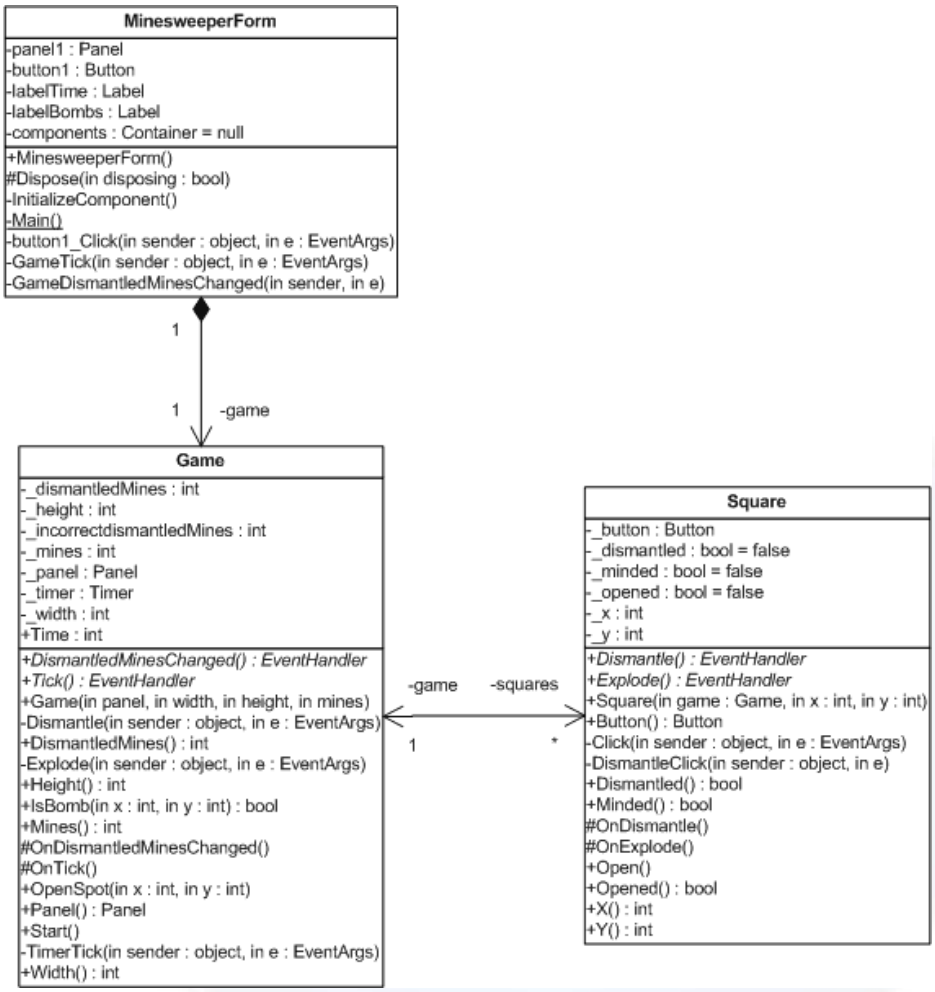
Jedan od najčešćih nedostataka tehničkih dokumentacija programskih rješenja je što često ne prati izmjene programa i nije ažurna. Kako je prikazano u poglavlju 4, postoje mnogi alati koji pomažu u izradi tehničke dokumentacije, omogućuju izradu složenih *UML* dijagrama i drugo. No, ukoliko se koriste posebni alati programer uvijek mora ručno voditi računa o ažuriranju pojedinih dijelova dokumentacije. Na primjer, ukoliko se napravi *UML* dijagram razreda pomoću alata, poput *Microsoft Visio*, programer mora ručno stvoriti cijeli model i održavati ga (unijeti ručno sve izmjene). Slika 9. prikazuje *UML* dijagram razreda koji je stvoren pomoću alata *Visio*. Dodatan nedostatak uporabe alata je što često nemaju sve tipove podataka koje imaju svi programski jezici. Samim time je teško stvoriti stvarnu sliku programskog rješenja koristeći takve alate.

Efikasniji način stvaranja tehničke dokumentacije je korištenje alata koji se nalate u razvojnim okruženjima (engl. *IDE – Integrated Development Environment*). Moderna razvojna okruženja sadrže velik broj alata koji mogu na temelju izvornog koda proizvesti razne *UML* dijagrame. Na taj način se svaka izmjena u programskom kodu odmah vidi u proizvedenoj dokumentaciji jer je ona napravljena na temelju koda. Dodatno, moguće je na temelju napravljenog *UML* dijagrama proizvesti dijelove programskog koda. Na primjer, definiranje funkcija i varijabli, ali implementaciju konkretnih koraka u pojedinim funkcijama, potrebno je i dalje ručno napraviti.

Neka popularna razvojna okruženja koja sadrže alate za izradu dokumentacije su:

- **NetBeans** – razvojno okruženje pisano u programskom jeziku Java. Podržava velik broj programskih jezika kao što su: *Java*, *JavaScript*, *PHP* (engl. *Hypertext Preprocessor*), *Python*, *Groovy*, *C/C++* i drugi. Ima veliku bazu dodataka koji se mogu dodatno preuzeti i instalirati. Dodatci obogaćuju osnovno razvojno okruženje novim funkcionalnostima, a postoje brojni dodaci koji se koriste za izradu dokumentacije ili pojedinih dijelova (npr. dijagram razreda).
- **Eclipse** – kao i *NetBeans*, ovo razvojno okruženje je napisano u programskom jeziku *Java* i podržava razvoj u svim popularnim programskim jezicima. Za razliku od okruženja *NetBeans*, otvorenog je koda i ima veliku zajednicu koja dodaje funkcionalnosti razvojnom okruženju. Preuzimanjem dodataka moguće je proizvesti razne dijelove dokumentacije.
- **Visual Studio** – razvojno okruženje organizacije *Microsoft* namijenjeno isključivo jezicima koji koriste *CLR* (engl. Common Language Runtime). Može se lako integrirati s *MS-SQL* bazom podataka i koristiti za proizvodnju tehničke dokumentacije.

- **InteliJ** – predstavljaju razvojno okruženje za programski jezik Java. Iako postoji besplatna inačica s manje funkcionalnosti, ovo razvojno okruženje je komercijalno. Besplatna inačica ima bitno manje funkcionalnosti i može se koristiti samo za proizvodnju dijagrama razreda. Ostale funkcionalnosti (kao proizvodnja dijagrama slijeda, paketa i drugih) su dostupne u punoj inačici.



Slika 9. UML dijagram razreda u alatu Visio

Izvor: http://www.reflectionit.nl/Articles/Minesweeper.aspx

3.3. Konvencije imenovanja u izradi programske potpore

Iako bi razni oblici programske dokumentacije trebali olakšati razumijevanje i zajednički rad na projektu, neke stvari uvijek ostaju skrivene u programskom kodu. Kako bi se olakšalo praćenje i razumijevanje programskog koda potrebno je sam programski kod kvalitetno oblikovati. Osnovno svojstvo svakog programskog koda je dobro imenovanje pojedinih entiteta i korištena konvencija imenovanja. Dobro imenovanje programskih varijabli, funkcija i ostalih entiteta u izvornom kodu utječe na njegovu čitljivost. Nazivi poput *proc1*, *proc2*, *tmp* čitatelju ne znače ništa i otežavaju analizu koda. Čak i naizgled dobra imena mogu biti dvosmislena. Na primjer, ukoliko funkcija kao rezultat vraća proste brojeve koristeći *Miller-Rabinov* algoritam moguće ime bi bilo *dajProstiBroj*. Iako daje određene informacije čitatelju, sam naziv nije dovoljno detaljan. Bolji naziv bi bio *millerRabinProstiBroj*, čime se jasno daje do znanja da se prilikom računanja prostog broja koristi *Miller-Rabinov* algoritam za provjeru prostih brojeva. Time se ujedno osigurava jednostavno proširenje programske potpore kada se, primjerice, žele dodati nove metode za dobivanje prostih brojeva.

Osim korištenja preciznih i jasnih naziva u imenovanju entiteta, poželjno je koristiti određene konvencije imenovanja. U prethodnom primjeru funkcije *mllerRabinProstiBroj* koristi se *CamelCase* notacija. Neke od popularnih programskih notacija su opisane u nastavku, a više o notacijama i dobrim praksama izrade koda moguće je pronaći u dodatnoj literaturi pod [5] i [15].

- **Mađarska notacija** (engl. *Hungarian notation*) – programske varijable i funkcije u imenu imaju oznaku povratnog tipa. Na primjer, *dbCijena* znači da funkcija vraća cijenu tipa *double.*
- **camelCase** – notacija u kojoj je prvo slovo svake riječi, osim prve, u nazivu veliko. Na primjer, funkcija za zapis u log datoteku može imati naziv *zapisiLogZapis.*
- **UpperCamelCase –** nastala po uzoru na *camelCase* notaciju. Prvi puta se pojavila s programskim jezikom *C#*, a nalaže da u nazivima funkcija sva početna slova riječi budu velika pa čak i prva. Na primjer, funkcija koja računa sumu dvaju brojeva bi mogla imati naziv *ZbrojiDvaBroja*. Za nazive varijabli se preporučuje običan *camelCase*.

3.4. ISO-IEC standardi za dokumentiranje programskih rješenja

Organizacija *ISO* (eng. *International Organization for Standardization*) je u 2009. godini objavila novu normu, *ISO/IEC 26513:2009 „Sustavi i softverski inženjering – Zahtjevi za ispitivačima i recenzentima korisničke dokumentacije“*. Ova međunarodna norma definira proces kroz koji se ispituju korisničke dokumentacije. Norma propisuje najmanje uvjete za ispitivanje i provjeru korisničke dokumentacije, uključujući tiskane i *on-screen* dokumente koje korisnici softvera koriste u radnom okruženju. To se odnosi na tiskane korisničke priručnike, *online* pomoć, vodiče (engl. *Tutorial*) i korištenje preporučene dokumentacije. Norma se bavi samo evaluacijom dokumentacije, a ne i evaluacijom programske potpore.

Norma ISO/IEC 26513:2009 razvijena je kako bi se olakšalo ispitivanje i pregledavanje korisničke dokumentacije kao dio procesa životnog ciklusa programske potpore. Standardom se mogu služiti ispitivači, recenzenti, analitičari i drugi članovi razvojnog tima koji su uključeni u dokumentiranje i održavanje projekta.

Procjena dokumentacije odvija se tokom same proizvodnje i održavanja, kao i kroz razne oblike provjera. Neke od njih su opisane u nastavku, a više informacija o standardima *ISO/IEC* može se pronaći u dodatnoj literaturi pod [9].

- **Recenzija dokumentacije** (engl. *Documentation Review*) – podrazumijeva pregledavanje korisničke dokumentacije programske potpore. Ovom provjerom se želi ustanoviti da li dokumentacija dovoljno jasno predočuje pojedine funkcionalnosti te da li je ažurna i dovoljno detaljna.
- **Provjera sustava** (engl. *System Testing*) – metoda provjere programske potpore kojom se provjerava čitav sustav kao cjelina. Metodom se provjerava da li razvijeni proizvod zadovoljava sve izrečene zahtjeve. Sam proces se izvodi kao provjera crne kutije (engl. Black Box Testing) jer se ne pregledava izvorni kod već samo gotova funkcionalnost.
- **Provjera uporabljivosti** (engl. *Usability Testing*) – zasniva se na stvarnoj uporabi programske potpore. Točnije, korisnici koriste sustav ili neki njegov dio i bilježe svoja zapažanja. Korisnička zapažanja se prosljeđuju razvojnom timu i na temelju njih se radi izmjena ili nadogradnja postojećeg sustava.
- **Provjera pristupačnosti** (engl. *Accessibility Testing*) – ovom provjerom se pregledava da li programsko rješenje pruža dovoljnu razinu pristupačnosti za pojedinu skupinu korisnika. Naime, da li programska potpora omogućava osobama sa smanjenim psiho-fizičkim sposobnostima normalan rad na sustavu.

Slika 10. Službeni logo organizacije ISO

Izvor: www.iso.org

4. Alati

Dokumentiranje programskih rješenja predstavlja opsežan zadatak koji sadrži mnogo ručnog rada (ažuriranje objektnog modela ili dijagrama baze podataka), a to se u projektima često preskače jer oduzima previše vremena. Kako bi se stvaranje i održavanje dokumentacije olakšalo razvijeni su brojni alati koji pomažu u stvaranju pojedinih dokumentacija. Većina alata pomažu u tehničkoj dokumentaciji i održavanju zahtjeva. Održavanje i stvaranje korisničke dokumentacije se zasniva na opisu pojedinih koraka i prikazom tih koraka putem slika. Kako je svaki sustav jedinstven, nije moguće napraviti alat koji će automatizirati stvaranje korisničke dokumentacije. U nastavku poglavlja se opisuju popularni alati za uređivanje zahtjeva te održavanje i stvaranje tehničke dokumentacije.

4.1. Alati za upravljanje zahtjevima

Složena programska rješenja sastavljena su od velikog broja međusobno isprepletenih zahtjeva. Ispravno razumijevanje i implementacija prikupljenih zahtjeva ključni su za uspjeh projekta. No, upravljanje zahtjevima je iznimno složen zadatak jer je teško unaprijed ustanoviti koji zahtjevi će uzrokovati konflikte te koliko će točno trajati implementacija. Jedan od najtežih prepreka u upravljanju zahtjevima je uočavanje međuovisnosti među zahtjevima. Kako bi se ovaj proces olakšao razvijeni su razni alati za upravljanje zahtjevima koji mogu uočiti poneke konfliktne zahtjeve i upozoriti korisnika. U nastavku poglavlja se opisuju neki od poznatijih alata za upravljanje zahtjevima.

4.1.1. Rational DOORS

Rational DOORS je komercijalni alat organizacije *IBM* za prikupljanje i održavanje zahtjeva. Stvoren je s ciljem olakšavanja procesa upravljanja zahtjevima prilikom razvoja programske potpore.

Alat pruža web sučelje putem kojeg je moguće pratiti razvoj zahtjeva. Svaku izmjenu zahtjeva moguće je napraviti pomoću jednostavnog korisničkog sučelja ili putem složenijeg sučelja koje omogućuje izmjenu dodatnih opcija. Sadrži podršku za format *Requirements Interchange* koji olakšava uključivanje klijenata i korisnika u razvoj programske potpore i analizu zahtjeva. Svakom članu je moguće ograničiti razinu pristupa kako bi se osigurala visoka razina sigurnosti i ispravnosti. Također, ima podršku za razne životne cikluse razvoja programske potpore te može prikazati utjecaj promjene pojedinih zahtjeva. Utjecaj promjene zahtjeva moguće je predvidjeti definiranjem međuovisnosti između pojedinih zahtjeva te njihovim grupiranjem. Čitav sustav je dostupan preko web sučelja koji olakšava pristup sustavu članovima razvojnog tima. Pristup putem web sučelja omogućava dodatak *Rational DOORS Web Access* koji podržava dvije uloge, urednika i recenzenta zahtjeva. Recenzent ima mogućnost pregledavana zahtjeva i komentiranja ukoliko postoje određene nesuglasnosti, a urednik ih može mijenjati. Više informacija može se pronaći na službenim stranicama programske potpore pod [12].

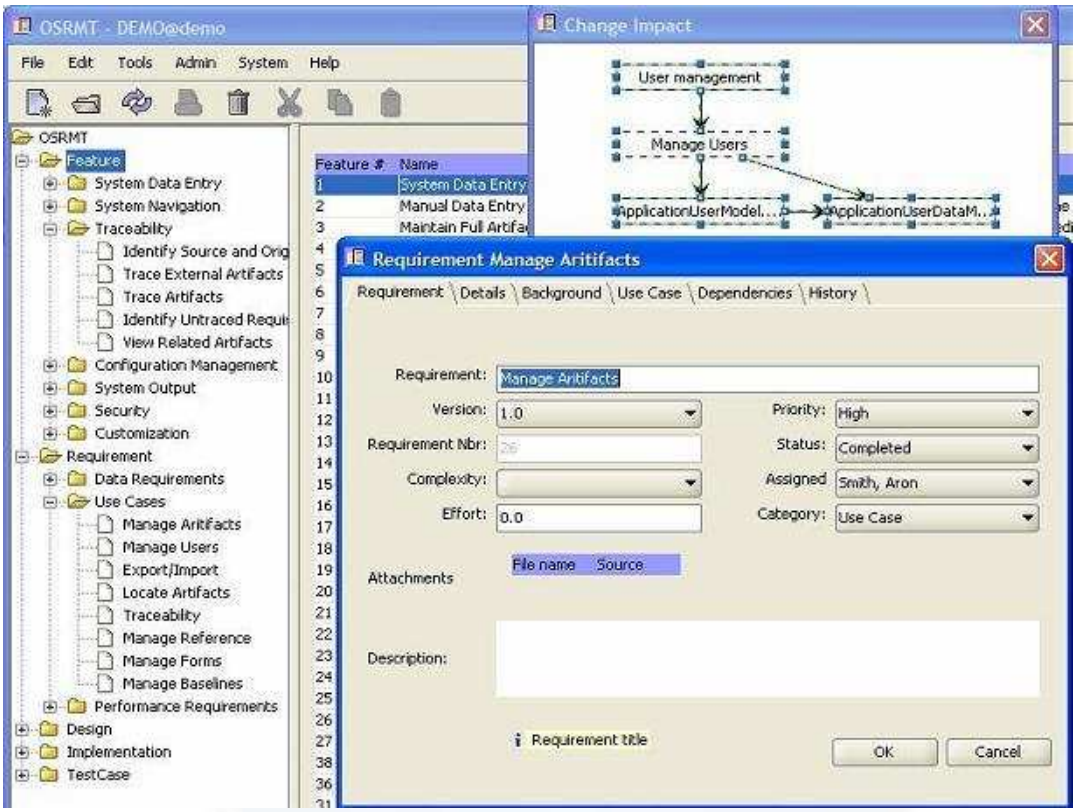
4.1.2. OSRMT

OSRMT (engl. *Open Source Requirements Management Tool*) je besplatan alat otvorenog koda namijenjen za održavanje zahtjeva. Alat ima veliku zajednicu koja ga održava i razvija dodatke koje je moguće preuzeti. U potpunoj je suglasnosti sa *SDLC*[4] (engl. *Software Development Life Cycle*) standardom za razvoj programske potpore. Podržava prikupljanje zahtjeva vezanih uz implementaciju, provjeru programske potpore, nove funkcionalnosti te sučelje i dizajn sustava. Podržano je i verzioniranje zahtijeva u skladu s promjenom inačice

[4] SDLC (engl. *Software Development Life Cycle*) – definira proces razvoja programske potpore. Tokom razvoja projekta prelazi se iz jedne faze životnog ciklusa u drugi, a ovisno o tipu projekta koristiti se drugačija metodologija razvoja. Metodologije imaju različite životne faze.

programske potpore. Alat se može koristiti u većim i manjim projektima neovisno o broju članova projektnog tima. Svakog korisnika je potrebno registrirati na sustavu, a prilikom pristupa potrebno je prijaviti se s vlastitm korisničkim podatcima. Korisnicima je moguće ograničiti pristup pojedinim skupinama zahtjeva ovisno o zaduženjima na projektu.

OSRMT je namijenjen operacijskim sustavim Windows i Linux/Unix, a napravljen je u programskom jeziku Java i zasniva se na arhitekturi klijent-poslužitelj. Postoje određeni preduvjeti za korištenje alata *OSRMT*. Na primjer, potrebno je imati Java inačicu 5.0 ili višu, pokretački program *JDBC* (engl. *Java Database Connectivity*) te pozadinsku bazu podataka. Zahvaljujući pokretačkom programu *JDBC* moguće je koristiti gotovo sve popularne baze podataka kao MySQL, Oracle, PostgreSQL i drugo. Slika 11. prikazuje korisničko sučelje alata *OSRMT*.



Slika 11. OSRMT alat

Izvor: http://download.cnet.com/OSRMT/3004-2383_4-10593125.html

4.2. Alati za tehničku dokumentaciju

Tehnička dokumentacija je bitan dio programske potpore koji je ključan u održavanju i nadogradnji projekta. Česta motivacija za izradu kvalitetne tehničke dokumentacije je njezina korisnost prilikom promjene jednog ili više članova razvojnog tima. Naime, s kvalitetnom tehničkom dokumentacijom novi razvojni inženjeri lako mogu uočiti posebnosti projekta i preuzeti ga. No, korisnost kvalitetne tehničke dokumentacije nadilazi potrebe novih članova razvojnog tima. Prilikom implementacije velikih i složenih sustava rješavaju se razni problemi i prepreke. Kada ne bi postojala kvalitetna dokumentacija bilo bi nezamislivo shvaćati i prisjećati se posebnosti pojedinih riješenih problema. Izrada kvalitetne tehničke dokumentacije je zahtjevan zadatak, a kako bi se olakšao razvijeni su razni alati. U nastavku poglavlja opisuju se neki od popularnijih alata za izradu tehničke dokumentacije.

4.2.1. Javadoc

Javadoc je alat za proizvodnju tehničke dokumentacije za sustave koji su pisani u programskom jeziku *Java*. Proizvodnja dokumentacije se zasniva na posebno oblikovanim komentarima u izvornom kodu. Moguće je proizvesti dokumentaciju u *HTML* obliku. Komentari moraju početi sa oznakom /** kako bi se prepoznao zapis. *Javadoc* komentari se uglavnom stavljaju iznad definicije funkcija, a koriste se posebne oznake za formatiranje dokumentacije. Neki od tih oznaka dane su u nastavku, a priložen je i primjer komentiranja koda s alatom *Javadoc*.

- **Author** – označava ime osobe koja je napisala zadanu funkciju.
- **Link** – stvara poveznicu na drugi odsječak dokumentacije. Korisno je kada jedna funkcija obavlja dio funkcionalnosti druge funkcije ili ako su po funkcionalnosti vrlo slične.
- **Parm** – oznake kojima se opisuju ulazni parametri. Moguće je opisati namjenu parametra, a tip se proizvodi samostalno na temelju funkcije,
- **Return** – ovom oznakom se opisuje povratna vrijednost funkcije. Kao i kod oznake *parm*, moguće je dodati kratki opis povratne vrijednosti, a tip će se proizvesti automatski.
- **Version** – pomoću ove oznake je moguće označiti inačicu pojedinog razreda. Ova oznaka se može pojaviti samo jednom u definiciji razreda.

```
/**
 * Provjerava je li šahovski potez dozvoljen.
 *
 * Koristiti {@link #doMove(int, int, int, int)} za obavljanje
poteza.
 *
 * @param theFromFile i-ta pozicija na kojoj se figura nalati
 * @param theFromRank j-ta pozicija na kojoj se figura nalati
 * @param theToFile   i-ta pozivija na koju figura ide
 * @param theToRank   j-ta pozivija na koju figura ide
 * @return            vrati TRUE ako je potez valjan, inače FALSE
 */
boolean isValidMove(int theFromFile, int theFromRank, int
theToFile, int theToRank)
{
    ...
}
```

4.2.2. Doxygen

Doxygen je alat za proizvodnju tehničke dokumentacije za programske jezike *C/C++*, *C#*, *Java*, *PHP*, *Python* i druge. Moguće ga je koristiti na svim operacijskim sustavima, a napravljen je u programskom jeziku *C++* i otvorenog je koda. Dokumentiranje se zasniva na pisanju posebno označenih komentara u izvornom kodu. Na temelju komentara se prozvodi dokumentacija u *HTML*, *PDF* ili *LaTeX* obliku koji je moguće uključiti u tehničkoj dokumentaciji kao opis funkcija. Isječak ispod daje primjer komentara napisanog u *Doxygen* obliku. Doxygen koristi određene oznake za detaljniji opis funkcija, a najvažnije oznake su u nastavku.

- **Author** – označava ime programera koji je zadužen za održavanje odgovarajućeg dijela programskog koda.
- **Parm** – označava ulazni parametar funkcije. Prilikom opisivanja parametra koristi se naziv parametra, njegov tip te kratki opis ukoliko je potrebno.

- **Returns** – označava povratnu vrijednost funkcije. Kao i kod oznake *parm*, moguće je definirati naziv i tip povratne vrijednosti te dodati kratki opis.

```
/**
* Brisanje pojma iz baze.
*
* Prilikom brisanja pojma iz baze ne izvodi se DELETE naredba već
se atribut
      * 'del' postavlja na 1 (što znači da je obrisan i bit će
isključen    iz daljnjih pretraga).
      *
      * @param $id
      *   Broj koji označava primarni ključ pojma u bazi podataka.
      */
     function deletePojam($id) {
           //Tijelo funkcije
}
```

4.2.3. Sandcastle

Sandcastle predstavlja alat za proizvodnju tehničke dokumentacije organizacije Microsoft za jezike platforme .NET. Kao i kod do sada spomenutih alata, dokumentacija se stvara na temelju posebno označenih komentara u izvornom kodu programa. Komentari se moraju nalaziti neposredno prije definicije funkcija ili klasa kako bi alat prepoznao gdje komentar pripada. Moguće je proizvesti dokumentaciju u HTML ili XML (engl. *EXtensible Markup Language*) obliku. U nastavku je primjer komentiranja izvornog koda, kao i popis bitnijih oznaka.

- **Returns** – ovom oznakom se označava povratna vrijednost funkcije. Moguće je opisati povratnu vrijednost funkcije.
- **Code** – označava početak programskog odsječka koji se ne prevodi. Točnije, ponekad se u komentarima uključuju konkretni odsječci programskog koda. Radi čitljivosti, ti odsječci se odvajaju u posebnim oznakama kako bi se naglasilo da se radi o primjeru.
- **Summary** – oznaka kojom se označava početak komentara *Sandcastle*. Neposredno nakon početne oznake slijedi opis funkcije ili klase, a zatim se ugnježđuju druge oznake kako je prikazano primjerom.

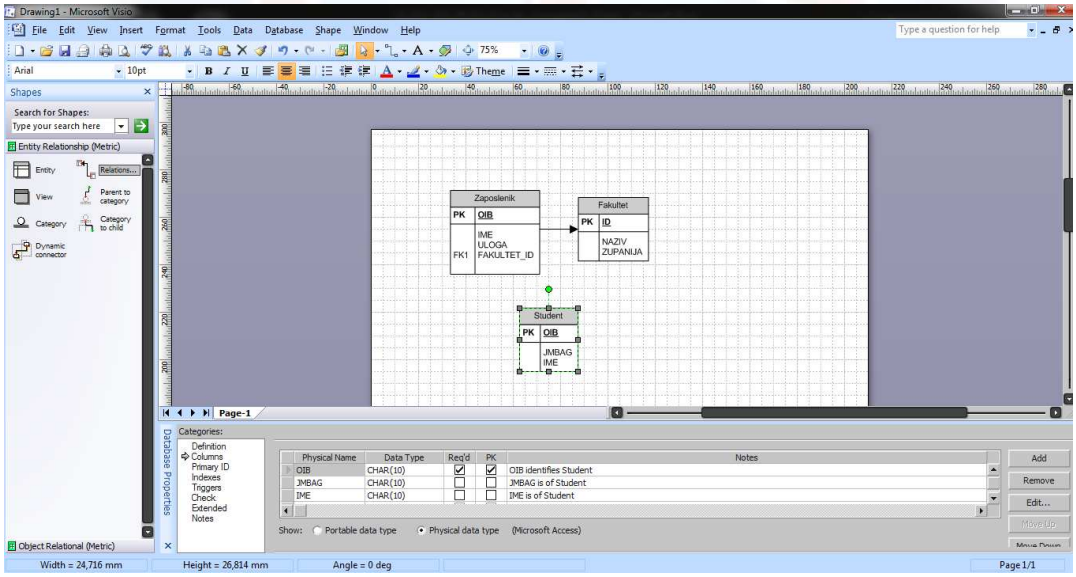
```
        /// <summary>
        /// Provjerava je li vrijednost jednaka NULL.
        /// </summary>
        /// <returns>
        ///     <c>true</c> ako je NULL; inače, <c>false</c>.
        /// </returns>
        public bool IsPropertyNull()
        {
           //Tijelo funkcije ...
       }
```

4.2.4. Microsoft Visio

Za razliku od prethodnih alata koji su se oslanjali na komentare u programskom kodu, *Microsoft Visio* se koristi za izradu različitih dijagrama. Podržani su razni tipovi dijagrama i moguće ih je pohraniti u obliku slike (npr. *PNG*, *JPEG*). *Visio* je jedan od alata programskog

paketa *Microsoft Office* i ima mogućnost integracije s ostalim alatima (kao *Word*, *Excel* i drugi). Podržani dijagrami nisu ograničeni na računarstvo već je podržana izrada dijagrama iz područja ekonomije, građevinarstva i slično. Kako je opisano u poglavlju 3.2, korištenje alata koji nije dio razvojnog okruženja za izradu dijagrama tehničke dokumentacije otežava njezino održavanje. Svaka izmjena u programu se mora ručno ažurirati u dokumentaciji. Ipak, ukoliko model ne postoji nije ga moguće prikazati s alatima razvojnog okruženja. Na primjer, u ranim fazama razvoja projekta korisno je imati okvirni model po kojemu se radi početna analiza zahtjeva i razrada daljnjih faza projekta. U tom slučaju se ističu alati kao *Visio* u kojima se dijagrami rade ručno. Neki od podržanih tipova dijagrama za dokumentiranje programskih rješenja su:

- **Dijagrami baza podataka** – baze podatka se vrlo često koriste u brojnim aplikacijama kao izvor i mjesto za perzistiranje informacija. Iz tog razlog je bitno dobro dokumentirati različite aspekte baze podataka (kako je opisano u poglavlju 2.2.1). Visio podržava različite dijagrame baza podataka, krenuvši od ER modela do dijagrama tablica.
- **UML dijagrami** – moguće je napraviti razne dijagrame koji su podržani UML standardom (opisan u poglavlju 3.1).
- **Dijagram toka podataka** – tok podataka često definira ponašanje programa, a najviše se koristi u proceduralnoj paradigmi izrade programske potpore. Razlog tomu je što su u proceduralnoj paradigmi funkcije vrlo usko vezane uz podatke i imaju strogo određeno ponašanje. U objektno orijentiranoj paradigmi podaci nisu toliko usko vezani za pojedine objekte budući da se u hijerarhiji nasljeđivanja često ponašanje može preusmjeriti na drugi razred.
- **Mrežni dijagrami** – omogućuju izradu raznih mrežnih topologija. *Visio* sadrži gotove ikone koje predstavljaju pojedine mrežne entitete kao što su poslužitelj, usmjeritelj, radna stanica i drugo.



Slika 12. Alat Visio
Izvor: LSS

5. Zaključak

Ovaj dokument opisuje problematiku dokumentiranja programskih rješenja. Radi se o procesu koji započinje puno prije samog programskog ostvarenja, ali se također nastavlja i nakon izrade konačnog proizvoda. U tom procesu potrebno je napisati više vrsta dokumentacije, a to su dokumentacija zahtjeva, tehnička dokumentacija i korisnička dokumentacija. Svaka od njih je namijenjena određenoj skupini ljudi (korisnicima, inženjerima, administratorima i drugim dionicima).

Prilikom izrade dokumentacije preporučljivo je koristiti neke od opisanih tehnike dokumentiranja i dobrih praksi. Neke od njih su UML dijagrami, konvencije imenovanja i korištenje razvojnog okruženja u održavanju dokumentacije. Kako bi se stvaranje i održavanje dokumentacije olakšalo, razvijeni su brojni alati koji pomažu u stvaranju pojedinih vrsta dokumenata. Većina alata pomaže u tehničkoj dokumentaciji i održavanju zahtjeva. Održavanje i stvaranje korisničke dokumentacije se zasniva na opisu pojedinih koraka i prikazom tih koraka putem slika. Kako je svaki sustav jedinstven, nije moguće napraviti alat koji će automatizirati stvaranje korisničke dokumentacije. Tehnička dokumentacija je bitan dio programske potpore koji je ključan u održavanju i nadogradnji projekta. Česta motivacija za izradu kvalitetne tehničke dokumentacije je njezina korisnost prilikom promjene jednog ili više članova razvojnog tima. Naime, s kvalitetnom tehničkom dokumentacijom novi razvojni inženjeri lako mogu uočiti posebnosti projekta i preuzeti ga. No, korisnost kvalitetne tehničke dokumentacije nadilazi potrebe novih članova razvojnog tima. Prilikom implementacije velikih i složenih sustava rješavaju se razni problemi i prepreke. Kada ne bi postojala kvalitetna dokumentacija bilo bi nezamislivo shvaćati i prisjećati se posebnosti pojedinih riješenih problema. Za razliku od tehničke dokumentacije, korisnička dokumentacija je namijenjena krajnjim korisnicima sustava. Korisnička dokumentacija ne sadrži objašnjenja programskog koda niti opis samog dizajna sustava. Njezin cilj je na jednostavan način objasniti kako koristiti pojedine funkcionalnosti programske potpore. Isto kao i tehnička dokumentacija, bitno je da korisnička dokumentacija bude uvijek ažurna i sinkronizirana s programskom potporom. Ukoliko se promjenom programa (ili zahtjeva) određen slijed akcija promijeni potrebno je odmah ažurirati korisničku dokumentaciju. Kada korisnička dokumentacija nije ažurna, korisnik neće moći koristiti programsku potporu što će uzrokovati nezadovoljstvo ili čak zastoj u radu.

Ovim dokumentom su prikazane prednosti u izradi kvalitetne programske dokumentacije. Također, dan je pregled nekih tipova dokumentacija te smjernice za izradu. Dokumentacija je ključan dio programske potpore, a njezino održavanje je jednako važno kao i održavanje samog programa.

6. Leksikon pojmova

CLR (Common Language Runtime)

Jezgrena komponenta okruženja Microsoft .NET. Predstavlja konkretnu implementaciju standarda CLI (engl. Common Language Infrastructure) koji definira okruženje za izvođenje programa. Putem sustava CLR, programski kod se prevodi u poseban oblik međukoda poznat kao jezik CIL (Common Intermediate Language). Programeri mogu koristiti bilo koji programski jezik koji je podržan infrastrukturom CLR za izradu svojih aplikacija u okruženju .NET.

http://msdn.microsoft.com/en-us/library/8bs2ecf4.aspx
http://whatis.techtarget.com/definition/0,,sid9_gci860097,00.html

JavaScript (Programski jezik JavaScript)

JavaScript je skriptni programski jezik, koji se izvodi u web pregledniku na strani korisnika. Napravljen je da bude sličan Javi, zbog lakšega korištenja, ali nije objektno orijentiran kao Java, već se temelji na prototipu i tu prestaje svaka povezanost s programskim jezikom Java. Izvorno ga je razvila tvrtka Netscape (www.netscape.com). JavaScript je izrađen primjenom standarda ECMAScript.
http://javascript.about.com/od/reference/p/javascript.htm
http://www.w3schools.com/js/default.asp

MD5 (Message-Digest 5 algoritam)

Jedan od najpopularnijnih *hashing* algoritama, korišten za generiranje sažetaka poruka. Kao izlaz daje 128-bitni sažetak dobiven miješanjem 512-bitnih blokova.

http://os2.zemris.fer.hr/algoritmi/hash/2002_fabris/index.htm

http://www.gohacking.com/2010/01/what-is-md5-hash-and-how-to-use-it.html

PHP (Hypertext Preprocessor)

Objektno-orijentiran programski jezik namijenjen prvenstveno za izradu dinamičnih web sjedišta. PHP je besplatan proizvod, objavljen pod licencom PHP License. Sintaksom je vrlo sličan popularnim jezicima poput C/C++, Java i Perl, a u potpunosti je implementiran u programskom jeziku C. Zbog jednostavnosti uporabe i visoke popularnosti postao je jednim od najpopularnijih jezika za izradu web sjedišta i usluga. Za razliku od jezika C/C++ i Jave koji su strogo tipizirani, PHP nema tipove podataka.

http://www.techrepublic.com/article/what-is-php/5074693

http://php.net/manual/en/index.php

http://www.php.net

XML (EXtensible Markup Language)

XML je kratica za EXtensible Markup Language, odnosno jezik za označavanje podataka. Ideja je bila stvoriti jedan jezik koji će biti jednostavno čitljiv i ljudima i računalnim programima. U XML-u se sadržaj uokviruje odgovarajućim oznakama koje ga opisuju i imaju poznato ili lako shvatljivo značenje.
http://webdesign.about.com/od/xml/a/aa091500a.htm
http://www.w3schools.com/xml/default.asp

http://www.w3.org/XML/

7. Reference

[1] J.T. Hackos, Managing Your Documentation Projects, Wiley, 1994.

[2] J.T. Hackos, Information Development: Managing Your Documentation Projects, Portfolio, and People, Wiley, 2006.

[3] A.S. Pringle, S.S. O'Keefe, Technical Writing 101: A Real-World Guide to Planning and Writing Technical Documentation, Second Edition, Scriptorium Press, 2003.

[4] R. Miles, K. Hamilton, Learning UML 2.0, O'Reilly Media, 2006.

[5] S. McConnell, Code Complete: A Practical Handbook of Software Construction, Microsoft Press, 2004.

[6] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford, Documenting Software Architectures: Views and Beyond, Addison-Wesley Professional, 2010.

[7] K. Wiegers, Software Requirements, Microsoft Press, 2003.

[8] D. Tuffley, Software User Documentation: A How To Guide for Project Staff, CreateSpace, 2011.

[9] S. K. Land, J. W. Walz, Practical Support for ISO 9001 Software Project Documentation: Using IEEE Software Engineering Standards, Wiley-IEEE Computer Society Pr, 2006.

[10] R.S. Smith, Writing a Requirements Document For Multimedia and Software Projects, CSU Center for Distributed Learning, 1997.

[11] M. Maguire, N. Bevan, User requirements analysis A review of supporting methods, Kluwer Academic Publishers, 2002.

[12] Rational DOORS - Features and benefits - Requirements management, IBM, http://www-01.ibm.com/software/awdtools/doors/features/requirements.html

[13] D. Knuth, Literate Programming, Center for the Study of Language and Information, 1992.

[14] E.W. Dijkstra, C.A.R. Hoare, O. Dahl, Structured Programming, Academic Press, 1972.

[15] A. Hunt, D. Thomas, The Pragmatic Programmer: From Journeyman to Master, Addison-Wesley Professional, 1999.