Dokumentiranje informacijske tehnologije

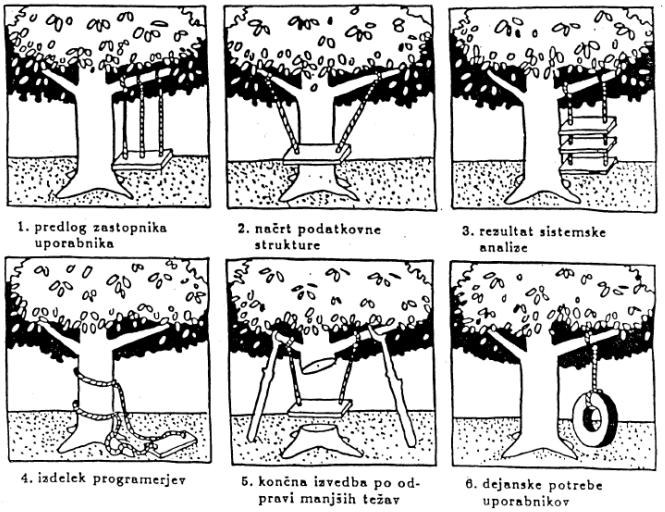
Peter Peer

peter.peer@fri.uni-lj.si
http://www.lrv.fri.uni-lj.si/~peterp

v0.9 9/2003

Skripta namenjena predavanju istoimenskega predmeta na
Višji strokovni šoli Šolskega centra Velenje, program Informatika.

Problemi razvoja sistemov #1



Osnovne ideje predstavitve snovi:

Brainstorming

Konstantna diskusija

Praksa

Ni znanost!

Pridiganje – niti dva projekta nista enaka!

Umetnost – Obrt – Industrija

Seminarji???

Problemi razvoja sistemov #2

V Fortranskem programu, ki je kontroliral vesoljsko sondo Mariner na poti k Veneri, je zgolj zamenjava vejice s piko povzročila njeno izgubo. Namesto `DO 3 I = 1,3` je v programu pisalo `DO 3 I = 1.3` in prevajalnik je namesto zanke, ki naj bi se trikrat izvedla, pripisal spremenljivki `DO3I` vrednost `1.3`.

Izhodišče za pripravo snovi

Zloženka & Katalog znanj – Informatika – VSŠ Velenje –

Predmetnik & tipična dela:

- Modul: Programske aplikacije in pod. baze
  1. **Načrtuje, pripravi, izvede in kontrolira lastno delo** in delo skupine
  6. **Vzdržuje dokumentacijo programskih rešitev in pripravlja uporabniška navodila**
- Modul: Računalniški sistemi in omrežja
  1. **Analizira obstoječe stanje, pripravlja rešitve za izboljšanje delovanja RSO ter sodeluje pri njihovi izvedbi**
  6. **Zbira, vodi, arhivira dokumentacijo o celotnem sistemu, delu, postopkih in nalogah**

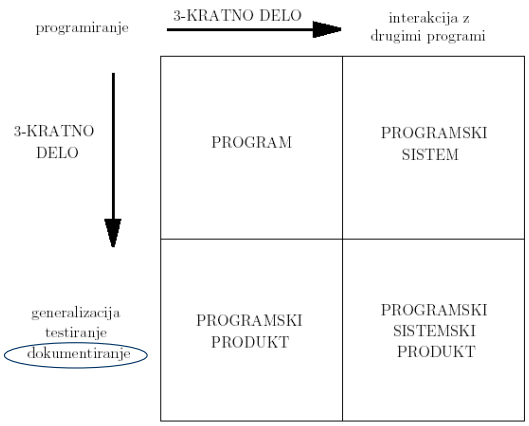
Oris predmeta

- Uvod v vodenje projektov
  - Programsko inženirstvo
  - Projektno delo
  - Osnovni razvojni modeli
  - Metode, ki sledijo načrtu
  - Agilne metode
- Dokumentiranje
  - Zagotavljanje varnosti
  - ISO 17799
  - **Razvojna dokumentacija**
  - **Uporabniška dokumentacija**

Plan-driven, Heavyweight, konvencionalne , planske metode?

Agile, Lightweight

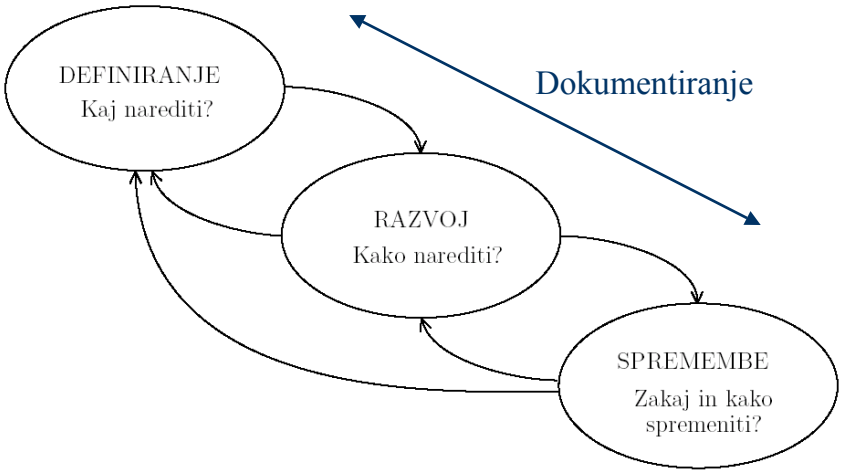
Programsko inženirstvo



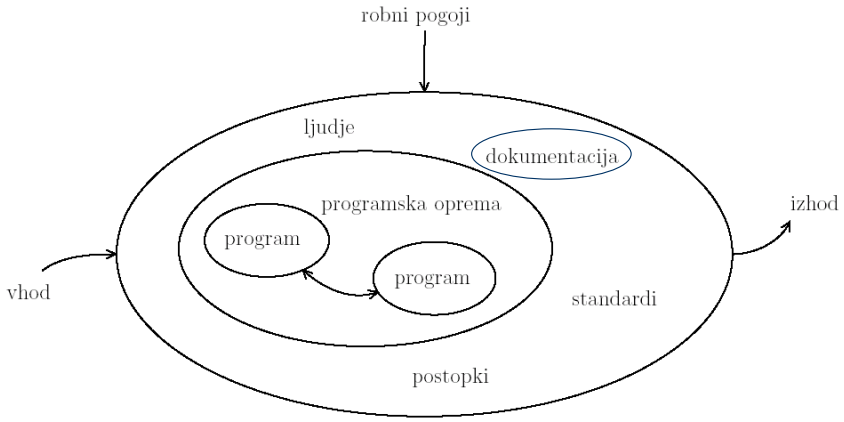
programsko inženirstvo
(10-KRATNO DELO)

Nadzor v smislu doseganja načrtovane kvalitete, porabljenega časa in stroškov je slabši v primerjavi z drugimi tehniškimi področji!

Generičen razvojni cikel



Informacijsko načrtovanje



Obseg programskega inženirstva

- specifikacija programske opreme (analiza: kaj?)
- načrtovanje programske opreme (kako?)
- programerske tehnike in orodja
- validacija programske opreme
- vodenje projektov

Specifikacija programske opreme

- definicija zahtev (Software Requirements Definition)
- modeliranje sistema (System Modelling)
- specifikacija zahtev (Requirements Specification)
- strukturirane metode (neformalne: SSADM, SASD,...)
- formalne metode (stroga matematična podlaga: VDM, Z sheme,...)
- prototipiranje (Software Prototyping)
- ...

O vsaki točki, podtočki,... je napisana knjiga(e)!

Načrtovanje programske opreme

- funkcijsko usmerjeno načrtovanje
- objektno usmerjeno načrtovanje
- načrtovanje sistemov v realnem času
- načrtovanje uporabniškega vmesnika
- ...

Programerske tehnike in orodja

- zanesljivost programske opreme
- ponovna uporaba programske opreme
- CASE (Computer Aided System Engineering) orodja
- okolja za razvoj programske opreme (Software Design Environment)
- ...

Validacija programske opreme

- verifikacija in validacija
- zanesljivost programske opreme
- varnost
- testiranje (ugotavljanje napak)
- orodja za testiranje
- statično preverjanje
- ...

Vodenje projektov

- izdelava planov
- tehnike mrežnega planiranja
- ocenjevanje stroškov in časa razvoja
- vzdrževanje programske opreme
- upravljanje s konfiguracijo (Configuration Management)
- **izdelava dokumentacije**
- zagotavljanje kakovosti
- ...

Projektno delo

Projekt je **enkratna,**
praviloma **zahtevna** in **kompleksna** skupina nalog,
ki mora biti končana v **določenem roku,**
doseči mora vnaprej določene in morebitne kasneje odkrite **cilje**
ter upoštevati **omejitve.**

Vrste projektov

- Deterministični (določnost) &
- Stohastični (empiričnost)

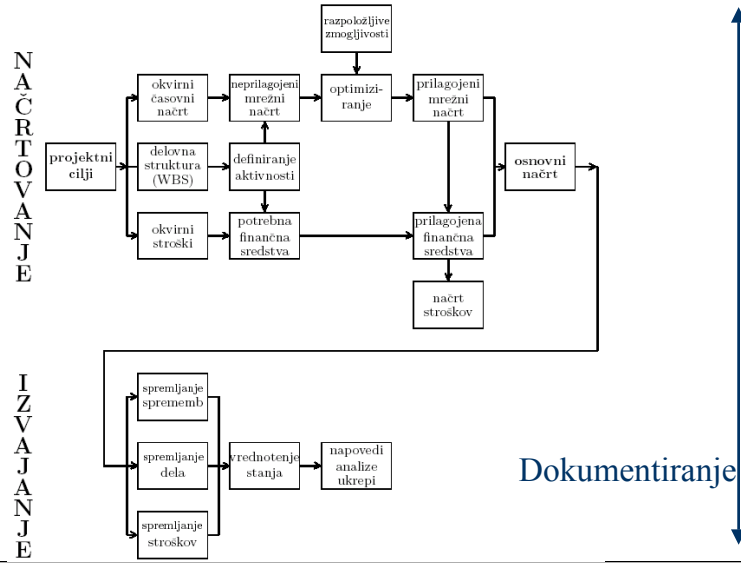
- Enkratni &
- Projektni procesi

Projektno : klasično upravljanje

- Upravljanje zajema načrtovanje, organiziranje, kadrovanje, vodenje in kontroliranje virov oziroma sredstev nekega podjetja, da bi dosegli finančne in druge zadane cilje tega podjetja.

- Klasično upravljanje: dolgoročnost in kontinuiranost dejavnosti
- Projektno upravljanje: specifično časovno obdobje in enkratni cilji

Delovne faze projekta



Skupina glavnega programerja

- Tako kot kirurg ali pilot tudi glavni programer sam opravlja vse bistvene naloge (analiza, načrtovanje, kodiranje), drugi člani ekipe pa mu pri tem asistirajo.
- Glavni asistent ga lahko kratkoročno nadomesti, tretji član skupine pa skrbi za administracijo in dokumentacijo.
- Skupini se lahko (občasno) pridružijo tudi drugi eksperti.
- Tehnična kompetentnost+voditeljske sposobnosti
- Pri velikih projektih v vlogi glavnega programerja nastopa skupina enakovrednih strokovnjakov, ki vodijo in nadzorujejo vse večje skupine programerjev.

vs. pair programming...

Ključne naloge projektne skupine

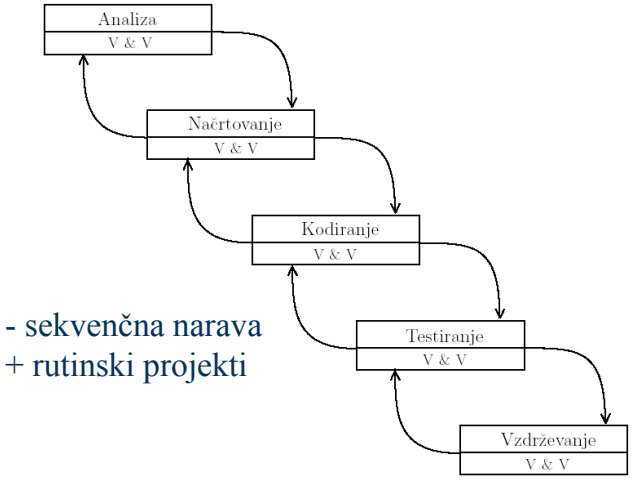
- podrobna in skrbna seznanitev z vsebino projektne naloge, z vsemi njenimi danostmi (standardi, glavni projekt, skupne informacijske osnove itd.), omejitvami in dokumentacijo ter drugimi informacijskimi viri, določenimi v odločbi;
- analiza naročnikovih zahtev, želja in pričakovanj kot izhodišče za določitev problemov in pričakovanih rešitev;
- izdelava vsebinske strukture in podrobnejšega načrta projekta z rokovnikom in imeni nosilcev posameznih aktivnosti ter s predračunom stroškov celotnega projekta;
- sestava okvirnega modela (podatkov in procesov) novega informacijskega sistema na podlagi popisa in analize obstoječih procesov, postopkov ter informacijskih in materialnih tokov (idejni projekt);
- izdelava projekta, njegova izvedba in prenos v prakso;
- seznanitev in usposobitev izvajalcev, uporabnikov in vzdrževalcev za izvajanje in vzdrževanje projekta;
- izročitev projekta v vzdrževanje z vso potrebno dokumentacijo;
- izdelava ocene o uresničitvi postavljenih ciljev, rokov in stroškov;
- usklajevanje dela z naročnikom ter seznanjanje le-tega s stanjem projekta in z uresničevanjem zastavljenih planov.

Osnovni razvojni modeli

- Klasični razvojni cikel
- Razvoj z uporabo prototipov
- Razvoj s 4. generacijo programskih jezikov
- Postopni razvoj programske opreme
- Spiralni razvoj programske opreme

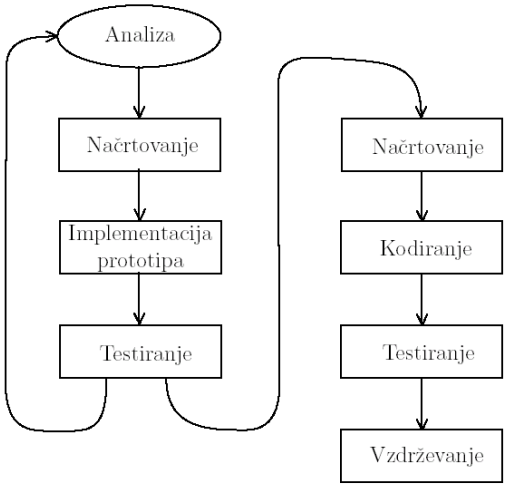
- Metode, ki sledijo načrtu
- Agilne metode

Klasični razvojni cikel



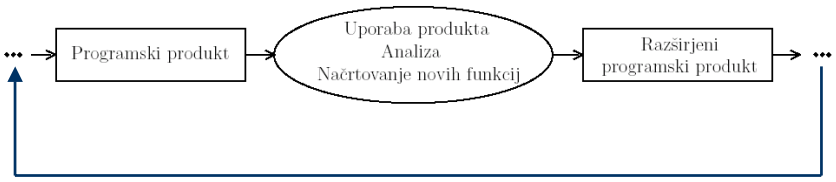
- sekvenčna narava
+ rutinski projekti

Razvoj z uporabo prototipov



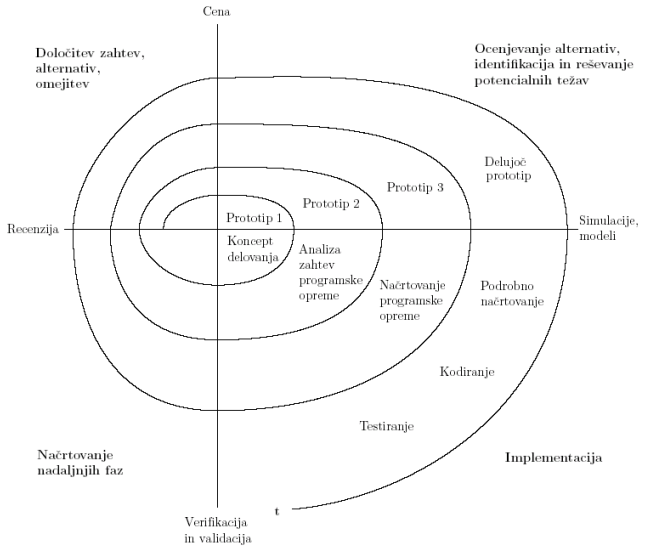
Kadar zahteve niso povsem jasno določene...

Postopni razvoj



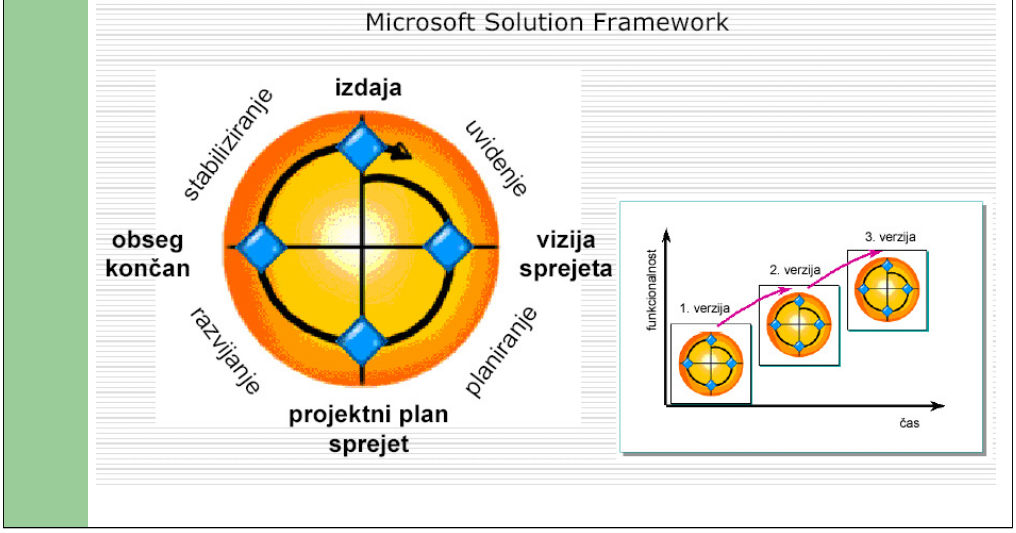
Redna uporaba

Spiralni razvoj



Hibrid

Primer iz prakse...



Primer procesnega ogrodja... (drug primer: RUP)

Metode, ki sledijo načrtu #1

- Primarni cilji: **napovedljivost**, **stabilnost**, **visoko jamstvo**
- Velikost: **velike skupine in projekti**
- Okolje: **stabilno**, **malo sprememb**, **osredotočenost na organiziranost in projekt**
- Naročnik: **interakcija po potrebi**, **osredotočenost na izvajanje pogodbe**
- Načrtovanje in nadzor: **dokumentirani načrti**, **numerična kontrola**
- Komunikacija: **eksplicitno dokumentirano znanje**

Metode, ki sledijo načrtu #2

- Tehnične zahteve: **formaliziran projekt**, **zmogljivost**, **vmesnik**, **kvaliteta**, **napoved evolucijskih zahtev**
- Razvoj: **obsežen načrt**, **dolgi inkrementi, preoblikovanje je privzeto kot draga naloga**
- Testiranje: **dokumentirani testni načrti in postopki**
- Razvijalci: **polovica** visoko usposobljenih na začetku, nato le **10%**, ostalo **programerji** z zmožnostjo sledenja metodi

Metode, ki sledijo načrtu #3

- Kultura dela: udobje in moč skozi ogrodje **načrtov** in **reda**

Agilne metode #1

- Primarni cilji: **hitro vidni rezultati**, **odzivnost na spremembe**
- Velikost: **manjše skupine in projekti**
- Okolje: **turbulentno**, **veliko sprememb**, **osredotočenost na projekt**
- Naročnik: **kolociran**, **osredotočenost na prioritetne inkremente**
- Načrtovanje in nadzor: **interni načrti**, **kvalitativna kontrola**
- Komunikacija: **zavedanje medosebnega (skupnega) znanja**

Agilne metode #2

- Tehnične zahteve: **prioritetizirane neformalne zgodbe in testi, predvidevanje novih sprememb**
- Razvoj: **enostaven načrt**, **kratki inkrementi**, **preoblikovanje je privzeto kot poceni naloga**
- Testiranje: **avtomatski testi definirajo zahteve**
- Razvijalci: **30%** visoko usposobljenih, ostalo **dobri programerji** z zmožnostjo sledenja metodi

Agilne metode #3

- Kultura dela: udobje in moč skozi ponujeno **svobodo** pri delu

Praksa

- Problem (neuspeh) **metod, ki sledijo načrtu**, je velika poraba časa in nefleksibilnost (spremembe) oziroma njena povezava s ceno
- V praksi se te metode redko dosledno uporabljajo (analiza=paraliza)
- Poudarek na izdelkih in procesih, bistveno manj na ljudeh

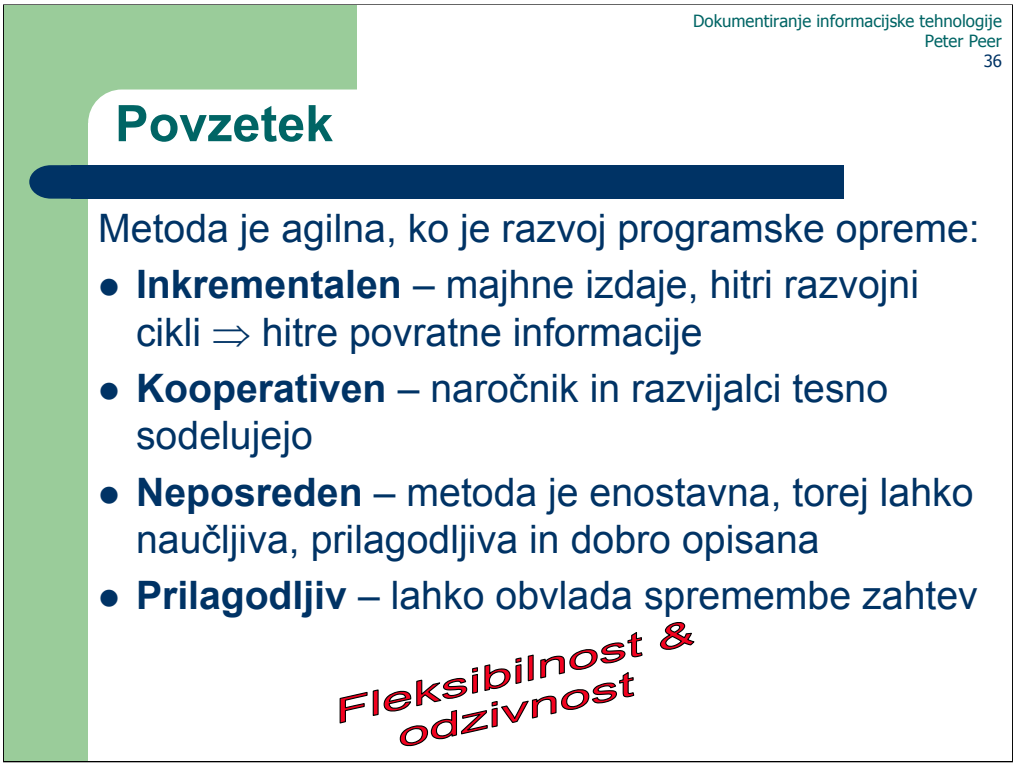
Statistika

- 40% projektov propade ali pa so opuščeni
- 80-90% programske opreme ne ustreza začrtanim ciljem
- 80% je dostavljene po izteku roka in prekorači proračun
- 25% pravilno integrira poslovne in tehnične cilje
- **Le 10-20% ustreza kriterijem uspeha**

Manifest za agilen razvoj

- **Posamezniki in interakcije med njimi** pomembneje od procesov in orodij
- **Delujoči programi** pomembneje od podrobne dokumentacije
- **Sodelovanje naročnika in razvijalcev** pomembneje od pogajanja o pogodbi
- **Pripravljenost na spremembe** pomembneje od sledenja načrtu

http://www.agilemanifesto.org



Cilj agilnosti: Zmanjšanje časa, stroškov in povečati fleksibilnost! Bistven je delujoč program!

Agilnost –

Ne v smislu:

•hitrosti razvoja na račun kakovosti

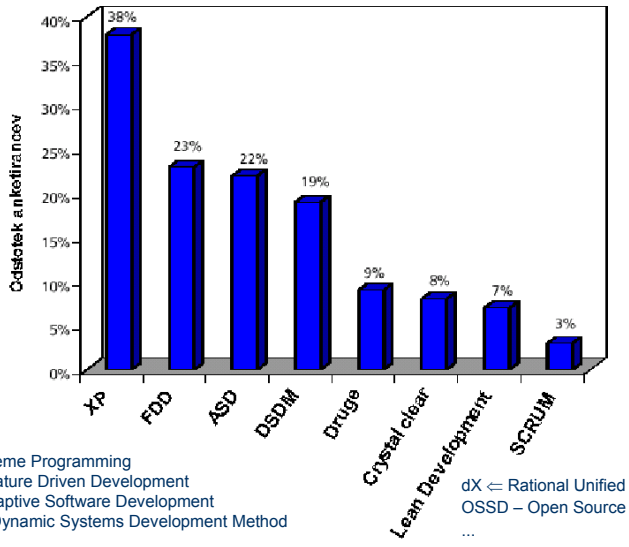
•minimalnega opisa procesa, ker bi potem bilo najbolje kar brez procesa

Temveč kot:

•Zmožnost SW organizacije, da se ustrezno prilagaja spremembam v okolju in zahtevam tega okolja

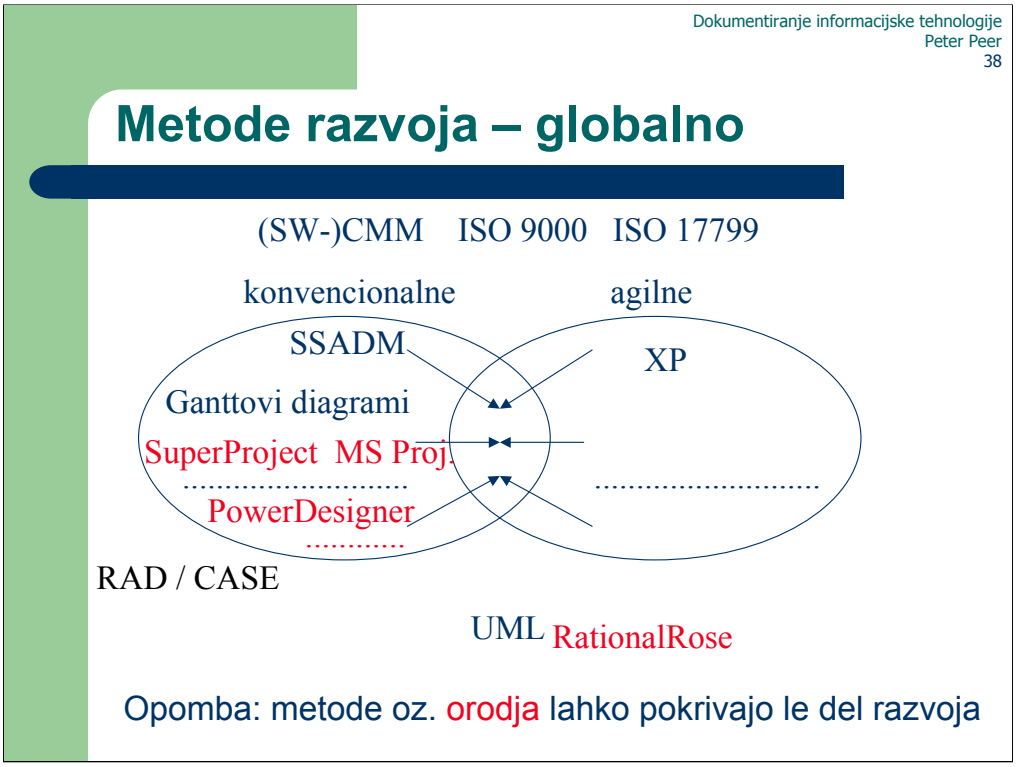
FLEKSIBILNOST, ODZIVNOST!

Agilne metode



XP – eXtreme Programming
FDD – Feature Driven Development
ASD – Adaptive Software Development
DSDM – Dynamic Systems Development Method

dX ⇐ Rational Unified Process (RUP)
OSSD – Open Source Software Development
...



Zgornjemu modelu in standardom poskušamo ustreči v postopku razvoja glede na primarne cilje, spodnji jezik (UML) pa uporabljamo v vseh (večini) fazah razvoja – nastajanje razvojne dokumentacije.

SW-CMM – Software Capability Maturity Model: Model opisuje zrelost razvoja programske opreme z namenom izboljšati zrelost razvoja v podjetju. Ima pet stopenj: začetna, ponovljiva, definirana, upravljiva in optimizirana.

UML - Unified Modeling Language: Modelirni (diagramski) jezik, ki je neodvisnosen od razvojnega procesa in uporabljenih programskih jezikov v implementaciji. Jezik UML v trenutni verziji definira notacijo in metamodel jezika. Notacija je grafična predstavitev jezika, sintaksa, ki sama po sebi ne daje jasnega in nedvoumnega pomena posameznih gradnikov. Večina objektnih metod ima v svoji definiciji le malo rigoroznosti, vendar so avtorji jezika za objektno modeliranje UML poskušali jeziku dati formalno podlago, brez 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.

Orodja, ki podpirajo UML: Oracle ODD, Platinum ParadigmPlus, Rational ROSE 98, Microsoft Visual Modeler,... Rational Objectory Process 4.1 ⇒ Rational Unified Process 5.0

Prikazane le nekatere (+PERT,...) metode, ki sem jih **sam** uporabljal:

Ganttovi diagrami,... – Mrežno načrtovanje: struktura projekta, čas, zmogljivosti, stroški

SSADM – Structured System Analysis and Design Method - Analiza zahtev (v osnovi) in načrtovanje IS

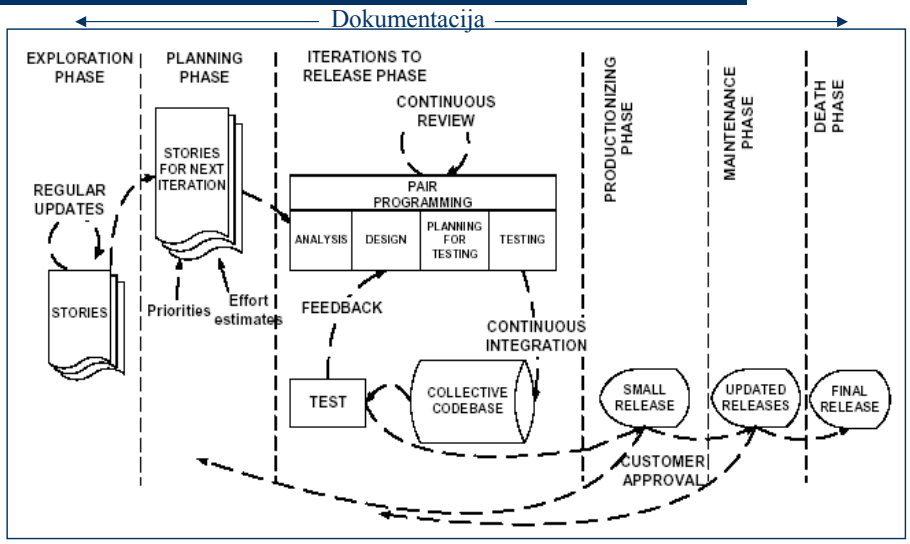
(+Heričko,...)

XP – eXtreme Programming

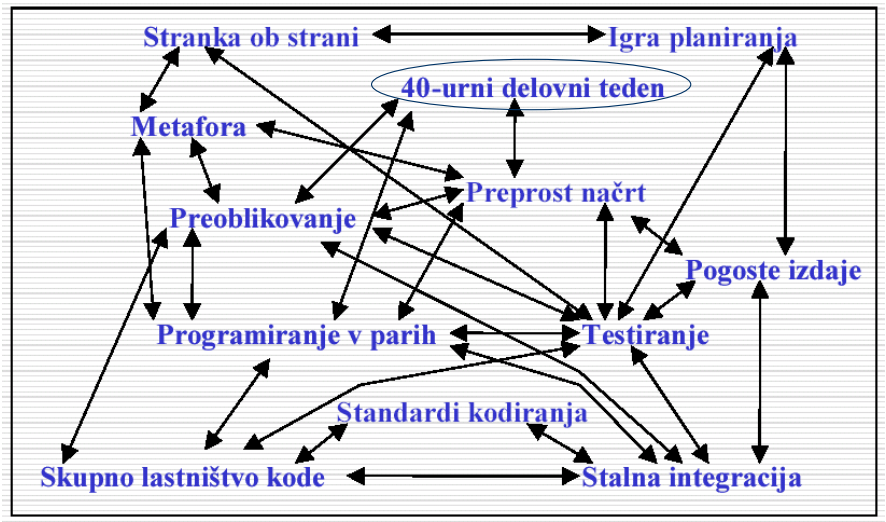
- Za majhne ali srednje velike skupine razvijalcev (med 2 in 20)
- Ob ohlapnih ali spreminjajočih se zahtevah
- Potreba in želja po hitri implementaciji poslovnih rešitev
- Najbolje deluje s sposobnim projektnim vodjem in skupino, ki je predana in izkušena in rešuje dobro definiran problem
- Ni za projekte, kjer je lahko ogroženo človeško življenje ali ključno premoženje

http://www.xprogramming.com

Življenski cikel XP procesa

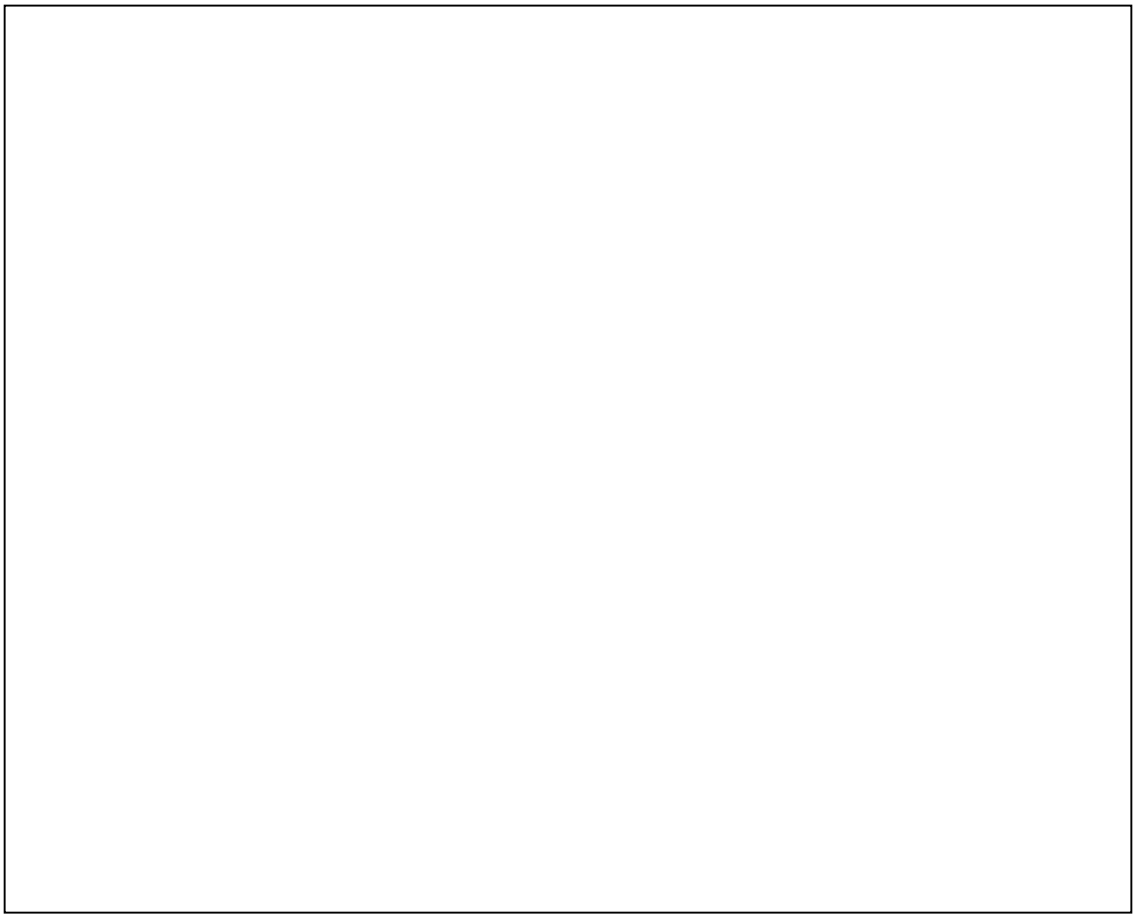


Principi XP


Stranka ob strani
Igra planiranja
40-urni delovni teden
Metafora
Preprost načrt
Preoblikovanje
Pogoste izdaje
Programiranje v parih
Testiranje
Standardi kodiranja
Skupno lastništvo kode
Stalna integracija

Uporabniške zgodbe

- Uporabniški primeri, ki kratko opišejo zahteve za sistem z naravnim jezikom
- Berljive za naročnika in razvijalca
- Ni nujno da so popolne; obljuba za nadalnje pogovore
- Za vsako značilnost (feature) trije stavki – kratek opis, ki ga v osnovi pripravi naročnik
- Nadomestilo za zahtevnike oz. dokumente zahtev v konvencionalnih metodah
- Razlika je v podrobnostih; ko nastopi ustrezen trenutek, razvijalec in naročnik sedeta skupaj in razširita osnovni opis
- Naročnik določi prioritete
- Tipični XP projekt, ki traja od 6 do 12 mesecev, zahteva od 50 do 100 uporabniških zgodb!



Planiranje iteracij

- Naročnik razvrsti uporabniške zgodbe glede na poslovno vrednost in izbere nekaj zgodb z najvišjo vrednostjo; podrobna določitev prioritet
- Razvijalci ocenijo potreben čas na osnovi: prejšnjih iteracij, hitrih prototipov, izkušenj
- Iteracije niso mejniki v časovnem planu!
- 4. spremenljivke: stroški, čas, obseg, kakovost
- "XP-merji" planirajo
- Igra planiranja (up. zgodbe na karticah): cilji (plan izdaj različic), elementi igre (up. zgodbe), igralci (naročnik, razvijalci) in poteze (premikanje kartic)
- Srečanja ob začetkih iteracij (izbira uporabniških zgodb za to iteracijo)

A project may be quantified by four variables:

•scope: how much is to be done

•resources: the people available

•time: deadline for the release or an iteration

•quality: how good the software is; how well it is tested.

The Management can make decisions about three of these four variables, but the fourth one depends on the Development. It should be noted, that lowering quality or hiring too many new people may have other negative impacts.

Poudarek na skupinskem delu

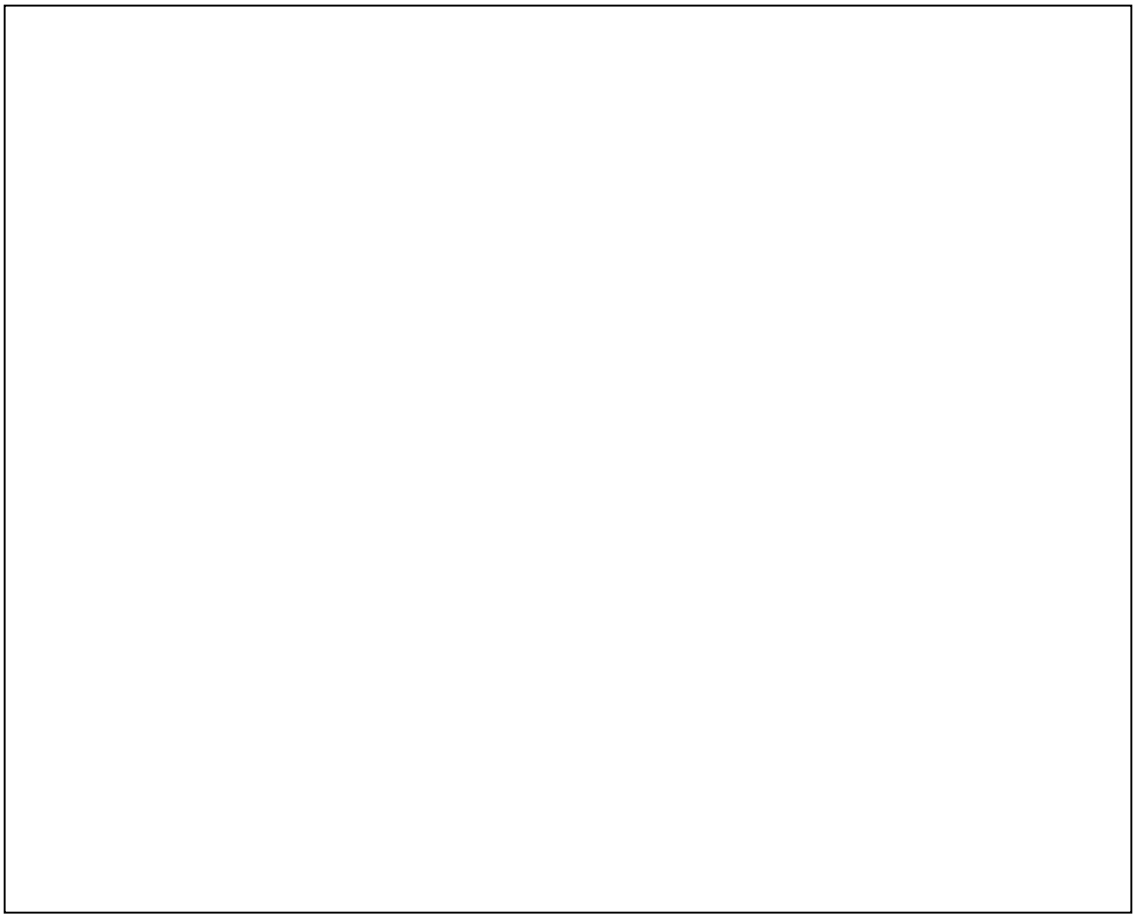
- Lastnik projekta je celotna skupina in vsak član lahko sodeluje na kateremkoli delu
- Zagovorniki XP trdijo, da je par več kot 2× produktivnejši kot posameznik
- Programiranje v parih (“pair programming”)

Pomembne so osnovne ideje metod, detajli pa morajo biti prilagodljivi vsaki skupini in projektu posebej! Te torej oblikujmo sami tako, kot to najbolje ustreza nasi skupini in nasemu projektu!!!

Programiranje v parih

- Ideja: dva programerja ki delata kot eden sta produktivnejša, kot če bi delala vsak zase, ker njun skupen izdelek vsebuje manj napak
- Vsa programska koda je razvita s pari razvijalcev; vsak par uporablja le en računalnik
- 2 vlogi: **voznik** in **navigator** – vlogi menjata
- Programiranje v paru ni enako sedenju pred zaslonom in opazovanju nekoga, ki tipka!!!
- Praksa??? Sočasno dokumentiranje?

http://www.pairprogramming.com



Testno voden razvoj

- **Najprej napišemo teste na osnovi zgodb, nato šele kodo, ki zgodbe realizira**
- Testiranje mora potekati **avtomatsko**
- Testi enote (Unit Tests – definirajo in izvedejo jih razvijalci)
- Funkcijski testi (Functional/Acceptance Tests – definirajo jih naročnik, izvedejo razvijalci)

- **Da napišemo teste, moramo razumeti zgodbe**
- Avtomatsko testiranje kot osnova za **preoblikovanje** (refactoring)

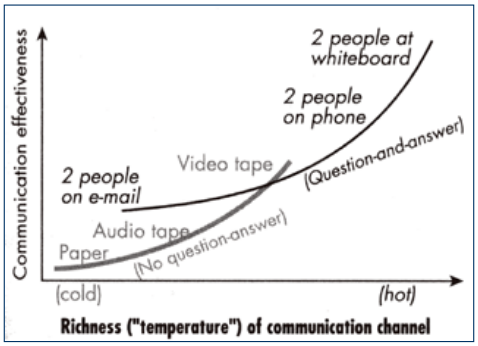
Preoblikovanje kode

- Preoblikovanje: spreminanje programske kode z namenom izboljšanja njene notranje strukture, brez sprememb zunanjega delovanja
- Preprečevanje entropije – slabšanje strukture programske kode
- Izvorna koda mora ostati preprosta in lahka za razumevanje
- **Najpreprostješa koda, ki uspešno opravi vse teste**
- **Koda je načrt in načrt je koda!**

The design documentation is maintained by the engineers within the code. Indeed, the code itself *is* the design documentation!

Osebna komunikacija > Dokumentacija

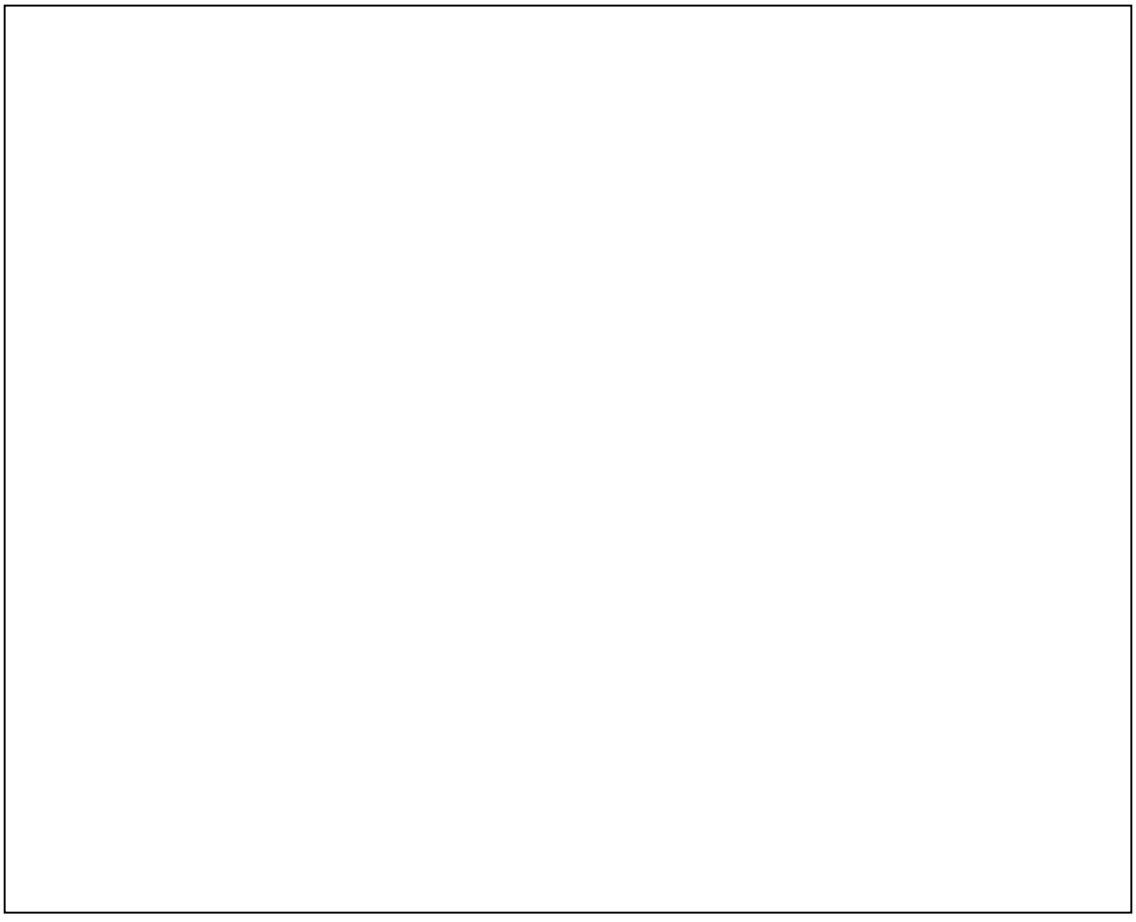
- Ne pomeni nič drugega kot omejeno količino dokumentacije
- Način dokumentiranja: table z avtomatskim zapisovanjem, video zapisi



Document at the end when the system architecture is stable. Use recording flipcharts, video.

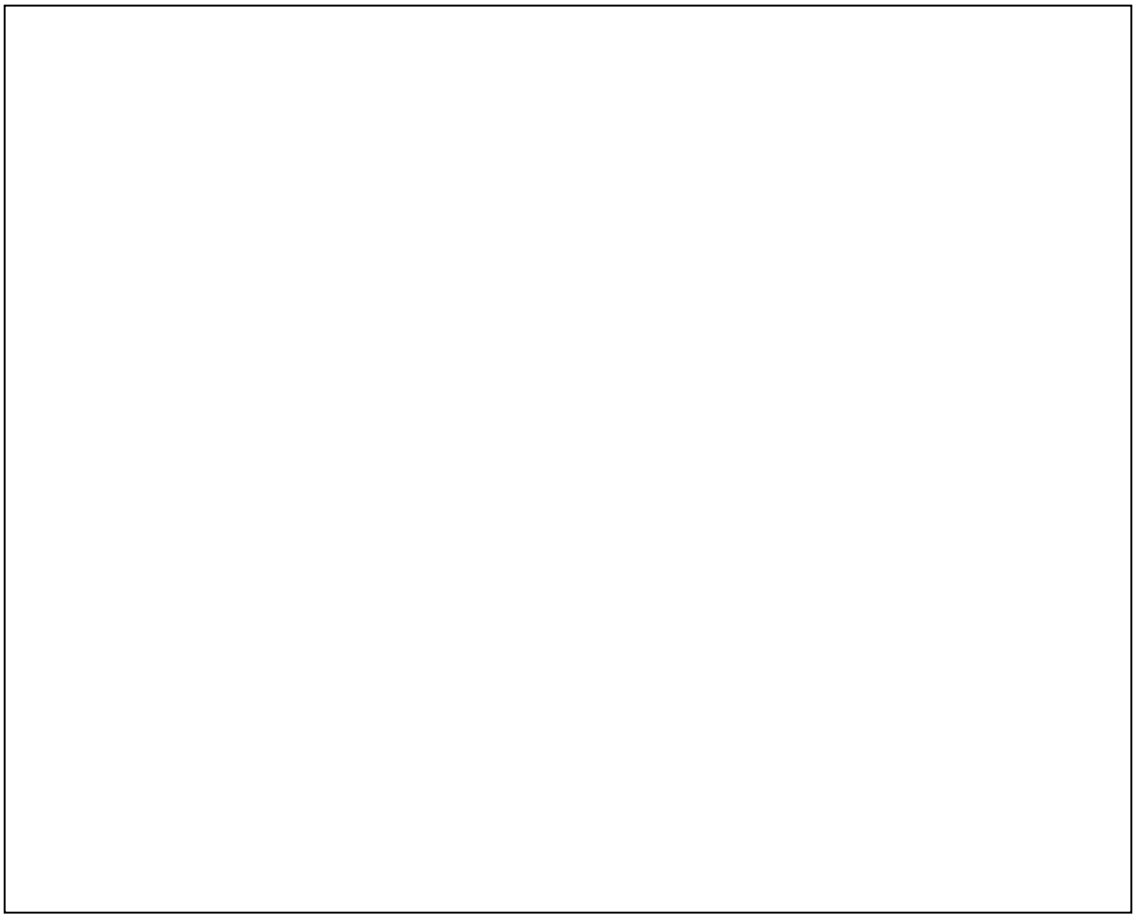
Delovno okolje

- Vsi v enem prostoru
- Že razporeditev v 2 nadstopji zmanjša komunikacijo
- Vsaj 17” ekran za programiranje v parih

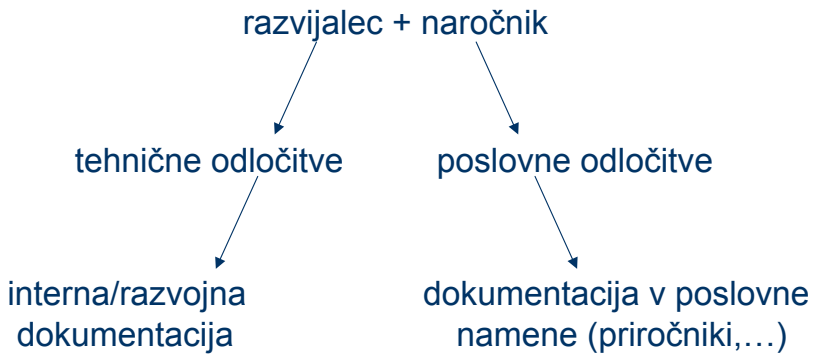


Skupno lastništvo kode

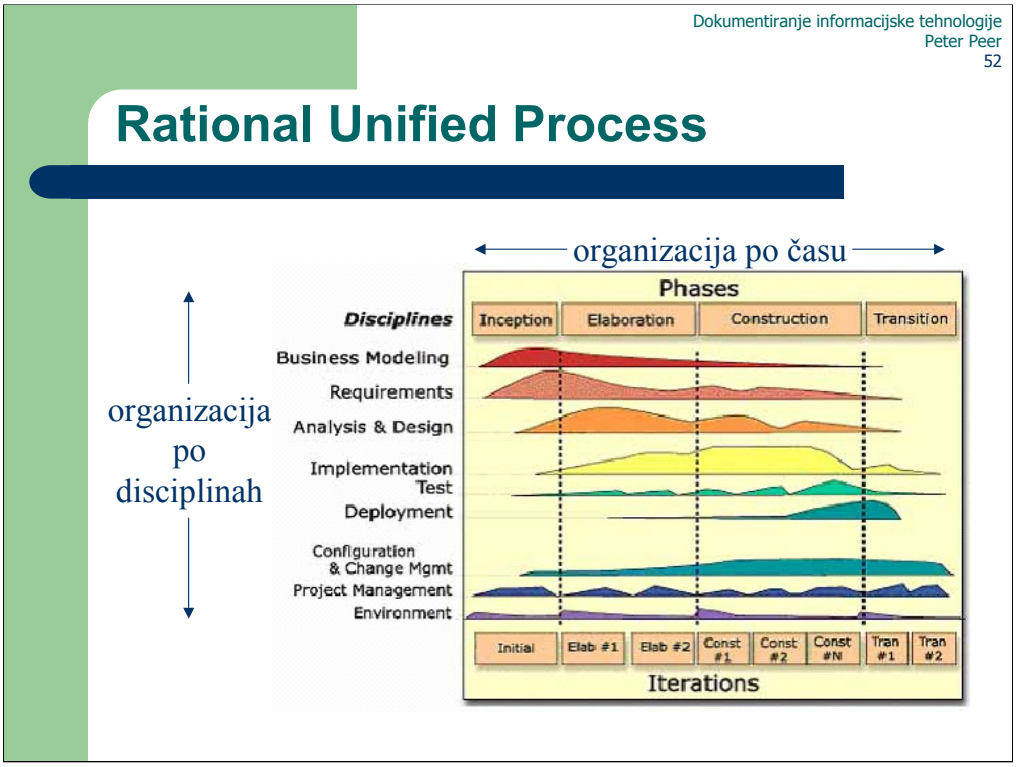
- Vsakdo lahko popravi katerikoli del kode
- Striktno upoštevanje dogovorjenih standardov kodiranja
- Rotiranje parov razvijalcev



Dokumentacija – vsakemu svoje



Odločitve o posamezni dokumentaciji so stvar prve ali druge skupine!



"The Rational Unified Process is the information systems methodolgy most widely in use today."

The Unified Process is a software development process, that is the set of activities needed to transform a users's requirements into a software system, but it is also seen as process framework, which can be specialised for different purposes.

Povedali smo že, da je jezik UML modelirni jezik in ne metoda razvoja, saj nima definiranega procesa razvoja, ki je pomemben del metode. Avtorji jezika pa so razvili tudi proces (metodo) razvoja RUP (Rational Unified Process), ki bi naj združevala najboljše postopke obstoječih procesov. Avtorji so že na samem začetku ugotovili, da splošnega procesa razvoja ne bo moč razviti, zato RUP predstavlja **ogrodje**, kjer si vsak uporabnik proces lahko **prilagodi** svojim zahtevam in uporabi samo tehnike, ki so primerne in smiselne za proces. RUP seveda izčrpno uporablja UML.

Iterativno inkrementalna narava objektnega pristopa ima za posledico prekrivanje med posameznimi fazami razvoja, kar narekuje predstavitev procesnega modela v dveh dimenzijah - po času (življenjski cikel procesa) in komponentah (disciplinah) procesa. Tovrstna predstavitev v procesnem modelu poveče dva ločena pogleda na razvoj programske opreme. Gledano iz časovne perspektive govorimo o razvojnih ciklih, fazah, iteracijah in mejnikih, kar je značilno za upravljalski nivo gledanja na projekt. Tehnično gledano je proces razvoja sestavljen iz komponent, aktivnosti, delovnih tokov, ipd.

Slika prikazuje dvodimenzionalni pogled na razvojni proces. Na sliki lahko vidimo, kako so posamezne faze organizirane po času in si sledijo zaporedno, hkrati pa je potrebno gledati tudi organizacijo po komponentah in delež posamezne konponente, ki se uporablja v posamezni fazi, kjer lahko vidimo, da se komponente praviloma uporabljajo v večih fazah.

Ali je RUP preobsežen?

- 9 disciplin
- Več kot 40 vlog (odgovornost, veščine,...)
- Stotine izdelkov (modeli, koda, dokumenti,...)
- Tisoče strani navodil (Kako izvesti neko aktivnost s pomočjo nekega orodja? – analiza=paraliza?)
- Potrebna prilagoditev
- V načrtu povežemo posameznike z vlogami, kar pomeni, da ni potrebno vsem poznati celotnega RUP-a.

Rational-ova orodja:

•Poslovno modeliranje: Requisite Pro, Rose, SoDA

•Zajemanje zahtev: Requisite Pro, Rose, SoDA

•Analiza in načrtovanje: Rose, SoDA, Apex

•Implementacija: Rose, Apex, SoDA, Purify

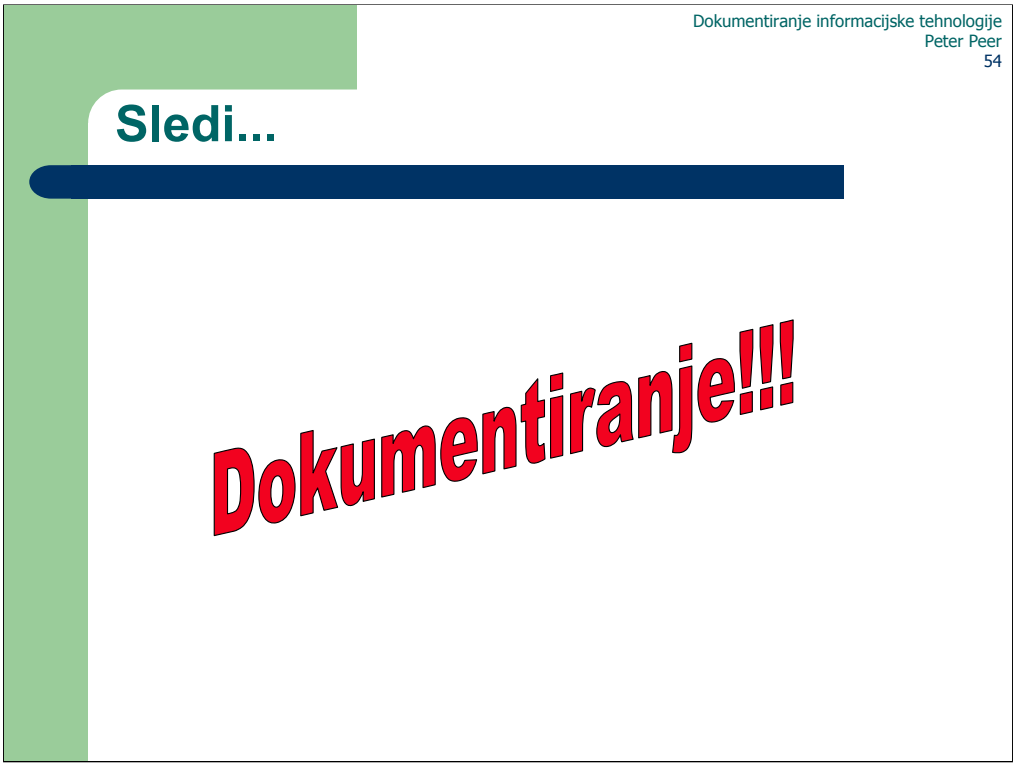
•Testiranje: SQA TeamTest, Quantify, PerformanceStudio

•Namestitev, dobava: SODA, ClearCase

•Upravljanje konfiguracije in sprememb: ClearCase, ClearQuest

•Upravljanje projekta: Unified Process, MS Project

•Upravljanje okolja: Unified Process, Process Workbench, RUP Builder



Ko slišimo RAZVOJ (osnovno načelo: iterativnost in inkrementalnost!), se moramo zavedati, da to vključuje tudi DOKUMENTIRANJE!!!

Documentation can be considered as a project in project!

Oris – ponovno, a podrobneje

- Zagotavljanje varnosti
  - Sistemska varnost
  - Zaupnost

  $\Rightarrow$
- ISO 17799 – povezava z dokumentiranjem

- **Razvojna dokumentacija**
  - UML
  - Dokumentiranje kode
- **Uporabniška dokumentacija**

  Pisanje brošur, specifikacij, priročnikov,...

Dokumentacija zagotavlja osnove varnosti – preglednost!

Dokumentitranje kode: XP – The design documentation is maintained by the engineers within the code. Indeed, the code itself *is* the design documentation!

Sistemska varnost

- Varovanje lastnih, partnerjevih in naročnikovih podatkov
- Transakcijska neoporečnost
- Ocena tveganja preko varnostne revizije
- Upravljanje tveganja (Risk Management)

- IBM Tivoli (orodje, velika skupina izdelkov)

E-poslovanje temelji na zaupanju. Podjetja morajo zaupati online storitvam, da zavarujejo tako svoje, partnerjeve in strankine podatke, kot tudi transakcijsko neoporečnost. Prvi korak je obširna ocena tveganja in varnostna revizija. Na žalost se varnosti ne da vključiti s preprostim stikalom. Zahteva pazljivo upravljanje tako računalniške infrastrukture kot tudi varnostnih naprav, upoštevajoč varnostno politiko podjetja.

V ta namen Tivoli (orodje, skupina izdelkov, IBM) nudi celovite rešitve, ki naslavljajo specifične varnostne rizike celotne informacijske infrastrukture; sistemi za kontrolo dostopa in upravljanje pravic, kot so infrastruktura javnih ključev (Tivoli Public Key Infrastructure), sistem enkratne prijave (Tivoli Global SignOn) in sistem dodeljevanja in vodenja pravic (Tivoli Identity Manager). V skladu z varnostno politiko podjetja Tivoli avtomatizirano upravlja in konfigurira vse varnostne vire (npr. požarni zid, strežnik za oddaljen dostop, itd: Tivoli Access Managers). Tivolijev pristop upošteva, da sta tudi nadzor in upravljanje infrastrukture pomembna varnostna elementa v podjetju povezanem v svetovni splet. Prepogosto se zgodi, da sistemski upravljalci, ki niso eksperti za varnost sistemov, upravljajo tudi te funkcije. V ta namen je Tivoli integriral funcije nadzora s sistemom varnostnega alarmiranja (Tivoli Risk Manager). Tivoli skrbno preiskuje sistemsko infrastrukto za varnostne luknje, ugotavlja vpliv na varnost ob vsaki sistemski spremembi ter jih sporoči upravljalcu. Tak celovit pristop omogoča IT organizacijam upravljanje celotne infrastrukture tudi s stališča varnosti.

Zaupnost podatkov in informacij

- Moralna odgovornost!!!
- “need to know basis”

ISO 17799

- Standard varovanja informacij (Information Security Standard)
- Cilj: vpeljali učinkovit sistem za upravljanje informacijske varnosti (Information Security Management System)
- BS 7799: ISO/IEC 17799 + BS 7799-2:2002

Kaj je standard?

Standard je:

•dokumentiran dogovor, ki vsebuje tehnične specifikacije ali druge natančne zahteve, ki naj bodo stalno uporabljane kot pravila oziroma smernice

•definicija karakteristik, ki zagotovijo skladnost materialov, proizvodov, procesov in storitev.

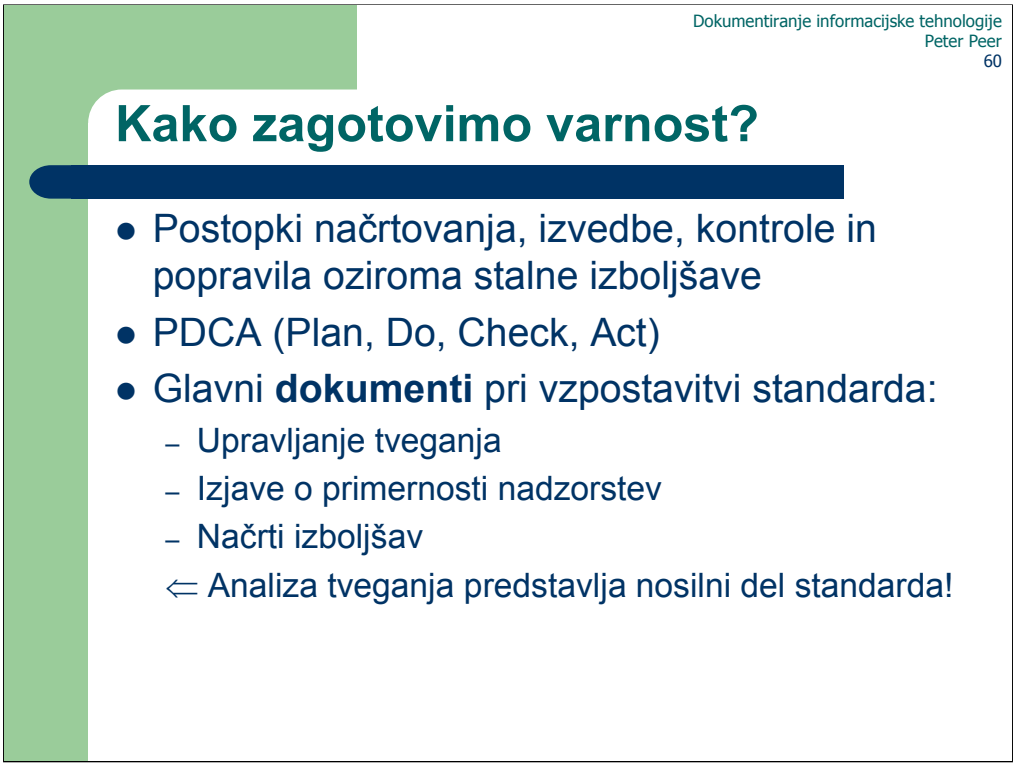
Omenjen standard obravnava varnost informacijskega sistema z različnih vidikov (fizična varnost, tehnološka kompatibilnost, tehnično varovanje premoženja informacijskega sistema, razvoj,...).

BSi (British Standards Institution) ima v lasti standard BS 7799, ki je bil sprejet kot ISO/IEC (International Standard Organisation/International Electrotechnical Commission) 17799 Code of practice for Information Security Management. Z BS 7799-2:2002 Specification for Information Security Management je bil standard nadgrajen in usklajen z nekaterimi ostalimi standardi.

BS 7799

Upoštevanje določil standarda BS 7799 nam omogoča, da bomo v podjetje vpeljali učinkovit sistem za upravljanje informacijske varnosti (ISMS), saj nam zagotavlja:

- ZAUPNOST – določena informacija je dostopna samo tistemu, ki mu je namenjena
- CELOVITOST – točnost in popolnost informacije ter metod obdelave le teh
- DOSTOPNOST – da imajo avtorizirani uporabniki dostop do informacije in s tem povezanih sredstev, kadarkoli je to potrebno



Kako zagotovimo upravljanje varnosti informacij?

Že pri doseganju kakovosti v manjših ali večjih sistemih oziroma organizacijah, se srečujemo z znanim postopkom načrtovanja, izvedbe, kontrole in popravila oziroma stalne izboljšave. Ta postopek je bolj znan pod kratico PDCA (Plan, Do, Check, Act). Pri vzpostavitvi standarda BS 7799/ISO 17799 so glavni dokumenti upravljanje tveganja, izjave o primernosti nadzorstev in načrti izboljšav, ki jih izvedemo iz tako imenovane analize tveganja. Analiza tveganja predstavlja nosilni del standarda.

Kontinuiteto in stalne izboljšave zagotavljamo z obveznimi notranjimi pregledi sistema. Zunanji neodvisni pregled pooblaščene ustanove pa nam omogoča tudi certificiranje skladnosti s standardom.

Iterative and incremantal Development (IID) grew from the 1930s work of Walter Shewhart, a quality expert at Bell Labs who proposed a series of short “plan-do-study-act” (PDSA) cycles for quality (ISO 9000) improvement.

...

Nadalje je bilo govora tudi o raznih organizacijah in posameznikih, ki v Sloveniji organizirajo razne seminarje, predstavitve, delavnice itd. ter se ob tem sklicujejo na standard BS 7799. Vse te zadeve bo preučila **pravna služba** BSi in v kratkem bo obiskal Slovenijo predstavnik te službe.

Razvojna dokumentacija in UML

- RUP uporablja UML skozi celoten proces
- Preobsežnost RUPa?

- Prilagoditev
- Uporaba UML pri načrtovanju na najvišjem (npr. zasnova) in najnižjem (npr. podatkovne strukture) nivoju?

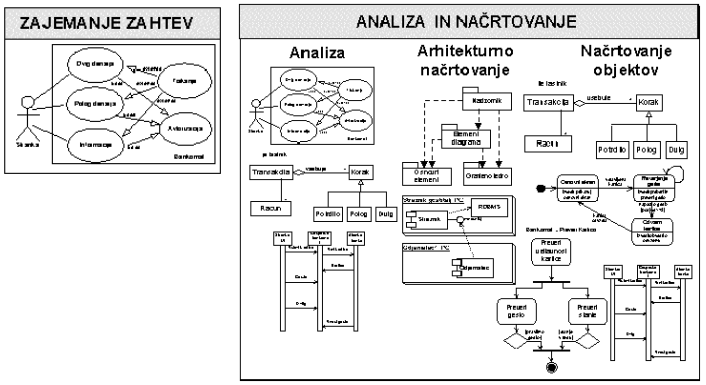
Unified Modeling Language

- Uporabniške zahteve (*diagram primerov uporabe*)
- Statična slika sistema (*razredni diagram*)
- Obnašanje programskega sistema (*diagram prehajanja stanj, diagram aktivnosti*)
- Interakcijo objektov (*diagram zaporedja, diagram sodelovanja*)
- Implementacijski konstrukti (*komponentni diagram, paketni diagram, diagram razvoja in dobave*)

Iz diagramskih tehnik jezika UML lahko razberemo **pet** različnih pogledov na problemsko področje. Prvi je pogled na **uporabniške zahteve**, kjer uporabljamo *diagram primerov uporabe*, ki izvira neposredno iz Jacobsonove metodologije in je bil za potrebe jezika UML poenostavljen. V drugo skupino lahko uvrstimo *razredni 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). Tretji pogled predstavlja skupina diagramskih tehnik, ki modelirajo **obnašanje programskega sistema**. Tu najdemo *diagram prehajanja stanj* (prevzet iz večih metod), s katerim opišemo prehajanja stanj za posamezne razrede in *diagram aktivnosti* (nov koncept), ki prikazuje izvajanje aktivnosti in dovoljuje tudi modeliranje paralelnih aktivnosti. Tretji pogled na programski sistem je pogled na **interakcijo objektov**, ki zagotavljajo neko funkcionalnost celotnega sistema, a smo ga zaradi večje preglednosti navedli kot samostojen (četrti) pogled na problemsko področje. V to skupino spadajo *diagram zaporedja* (prevzet iz večih metod, poimenovanje je v vsaki metodi praviloma drugačno), ki prikazuje tipično interakcijo objektov v scenarijih uporabe sistema in *diagram sodelovanja*, ki prikazuje način sodelovanja med množico objektov z namenom izvršiti določeno operacijo. Zadnjo, peto, skupino tvorijo diagrami implementacije, s katerimi prikažemo **implementacijske konstrukte**. V jeziku UML so to *komponentni diagram* in *paketni diagram*, ki prikazuje organizacijo komponent in paketov in *diagram razvoja in dobave*, s katerim prikažemo konfiguracijo programskega sistema in okolja, kamor sistem namestimo.

Veliko število diagramskih tehnik bi kaj hitro lahko uporabnika zmedlo in ga usmerilo v uporabo tehnologije zaradi tehnologije. Namen jezika UML je uporabiti le tiste diagramske tehnike, ki so pomembne za določeno problemsko področje, nikakor pa ne vseh tehnik na vseh področjih. Tipično uporabnost posameznih tehnik v procesu razvoja shematsko prikazuje naslednja prosojnica, kjer pa se moramo zavedati, da so diagrami prikazani le v fazah, kjer so posebno pomembni.

Tipična uporaba posameznih diagramov



primeri posameznih diagramov ⇒

Diagramske tehnike UML:

•diagram primerov uporabe (use case diagram);
Diagram primera uporabe predstavlja komunikacijo med uporabniki in računalniškim sistemom. Osnovni gradniki diagrama primera uporabe so: primeri uporabe, akterji, relacija (povezava) med primeri uporabe.

•razredni diagram (class diagram);
Diagrami razredov predstavljajo statično strukturo modela kot so razredi, relacije,… Ne prikazujejo dinamičnih informacij oziroma stvari, ki opisujejo časovno obnašanje. Diagrami objektov prikazujejo primerke, ki so skladni z diagrami razredov.

•diagram prehajanja stanj (state diagram);
Diagrami stanj so znana tehnika za opisovanje obnašanja sistema. Prikazujejo zaporedje stanj, skozi katere gre objekt v času obstajanja, ter dogodke, ki prožijo prehode med stanji.

•diagram aktivnosti (activity diagram);
Namenjeni modeliranju in prenovi poslovnih sistemov. Opisujejo potek dela oziroma korake, ki jih uporabniki počno pri svojem delu. Ključni pomen diagramov aktivnosti je, da omogoča poiskati paralelne procese.

•diagram zaporedja (sequence diagram);
Prikazuje časovno (zaporedno) sodelovanje objekta v interakciji, ne prikazujejo pa asociacije med objekti.

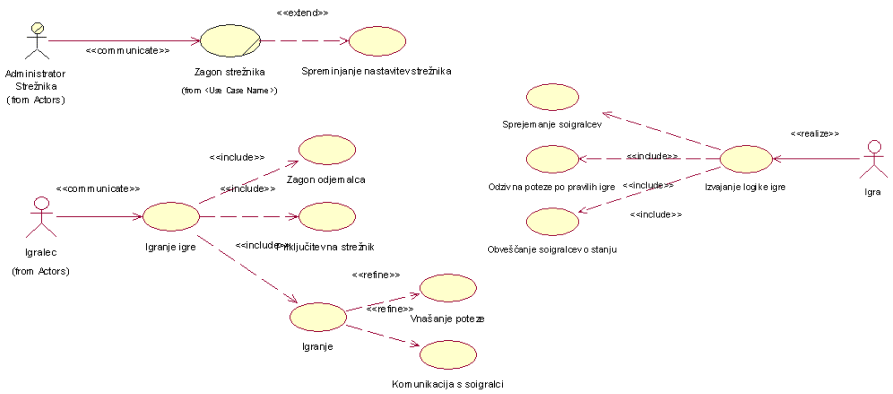
•diagram sodelovanja (interaction diagram);
Prikazujejo obnašanje posameznega primera uporabe ter sodelovanje objektov v sklopu enega primera uporabe.

•paketni diagram (package diagram);
Predstavljajo skupine razredov,... in njihove odvisnosti (povezave) med seboj.

•komponentni diagram (component diagram);
Predstavljajo programske komponente in njihove odvisnosti (povezave) med seboj.

•diagram razvoja in dobave (deployment diagram);
Prikazuje fizične zveze (relacije) med programskimi in strojnimi komponentami sistema.

Diagram primerov uporabe



Ozadje diagramov:

V nadaljevanju so predstavljeni primeri razvoja komponent potrebnih za mrežno igranje igre v standardni predstavitvi UML. Komponente so zasnovane z orodjem Rational Rose Enterprise Edition in sledijo smernicam za razvoj po metodi Rational Unified Process. Opisana mrežna igra je izvedena z Visual C++.

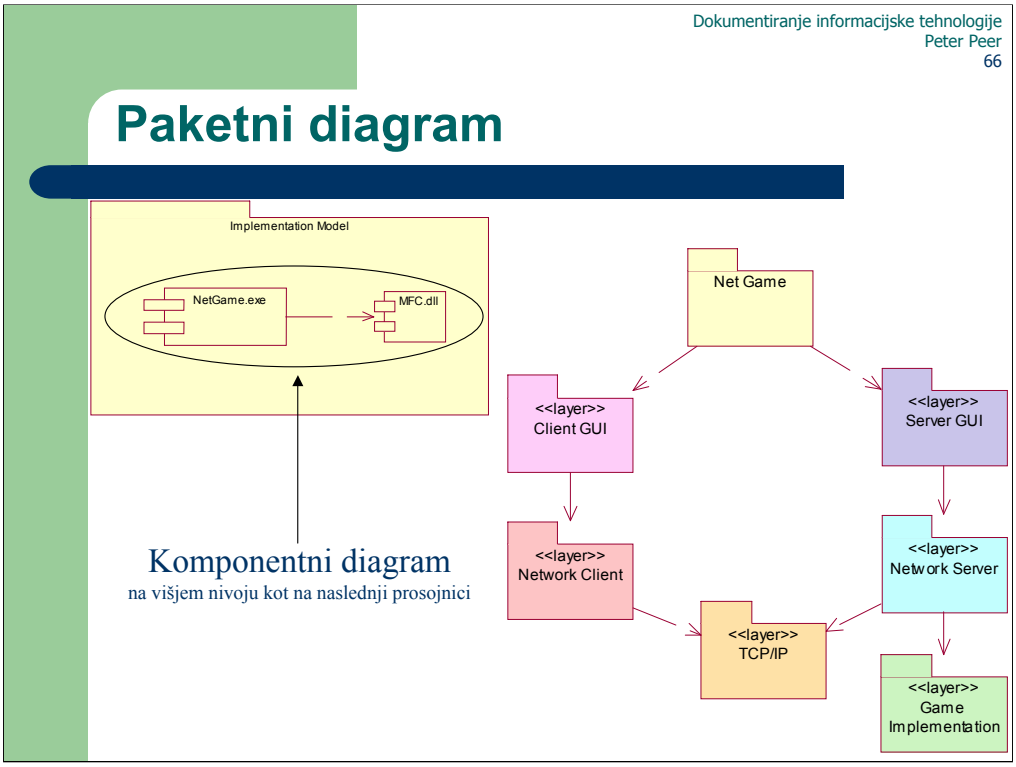
SVG

Matej T.

15 sekund slave – Borut

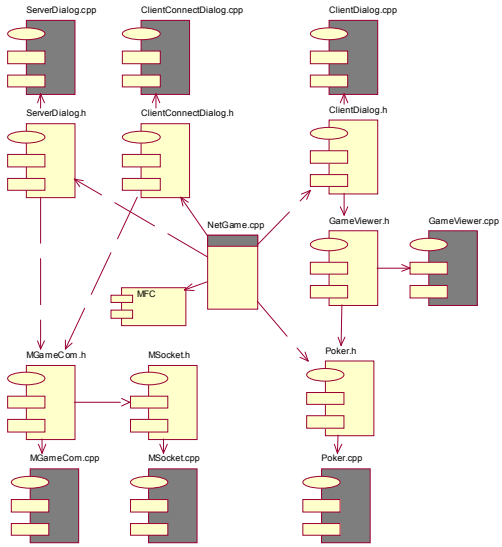
???

http://www.modelingstyle.info

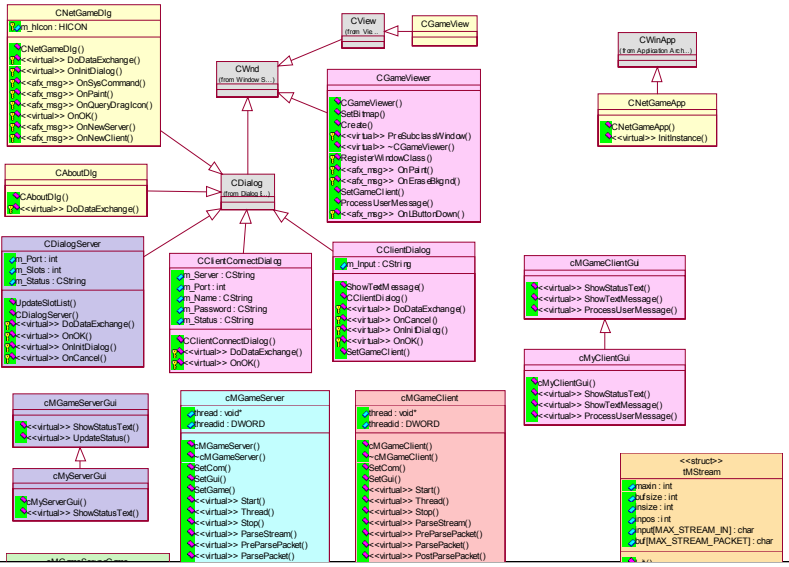


Desni diagram predstavlja konceptualno ogrodje za našo internetno igro. – Barvno kodiranje je uporabljeno v nadaljevanju za označbo pripadnosti razredov konceptualnim nivojem.

Komponentni diagram



Razredni diagram



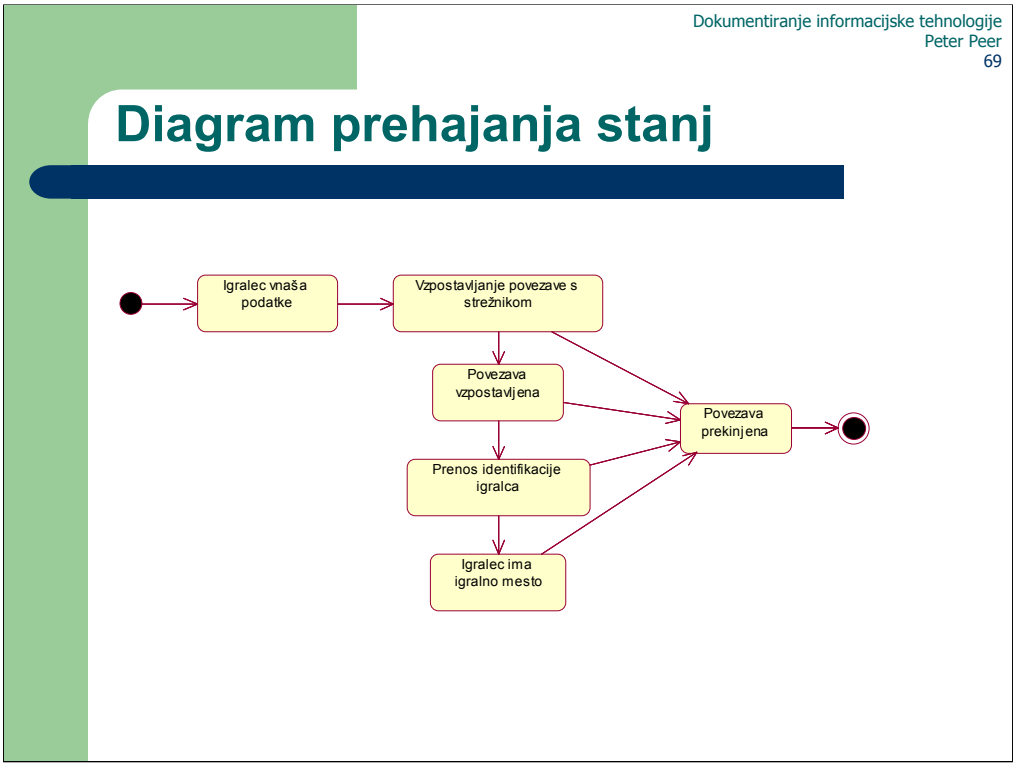
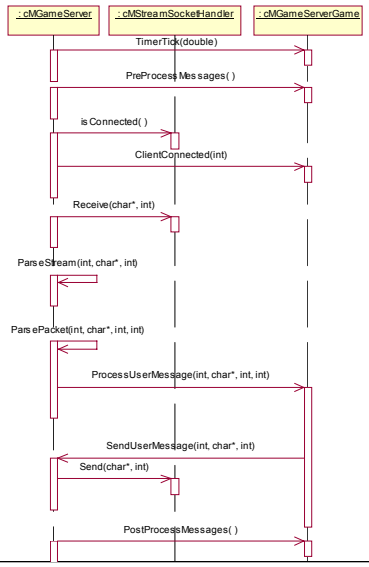


Diagram prehajanja stanj odjemalčevega priklopa na strežnik.

Diagram zaporedja



Glavna zanka periodično preverja stanje in bere pakete iz mrežnega vmesnika posameznih odjemalcev, jih posreduje virtualnemu razredu, ki izvaja logiko igre, leta pa vrača pakete z odzivi preko strežniškega razreda nazaj na odjemalčev vmesnik.

Diagram sodelovanja

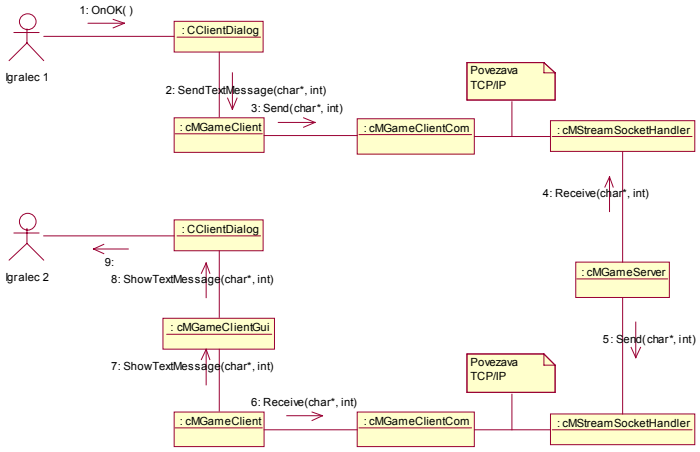


Diagram sodelovanja med razredi: igralec komunicira s soigralci po IRC principu.

Diagram razvoja in dobave

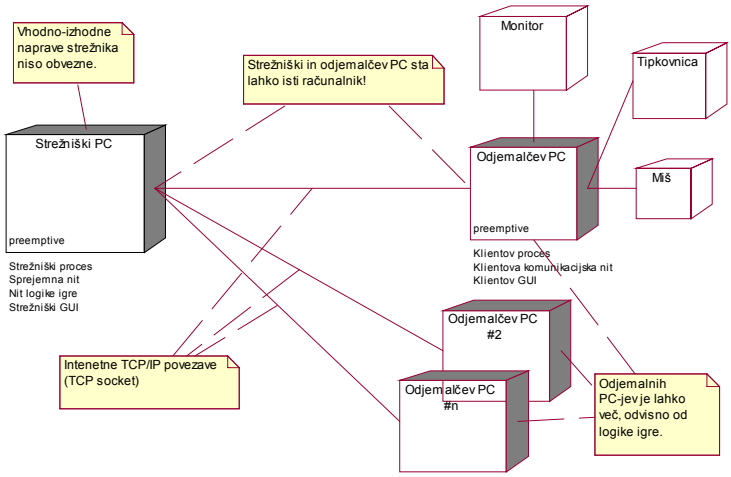
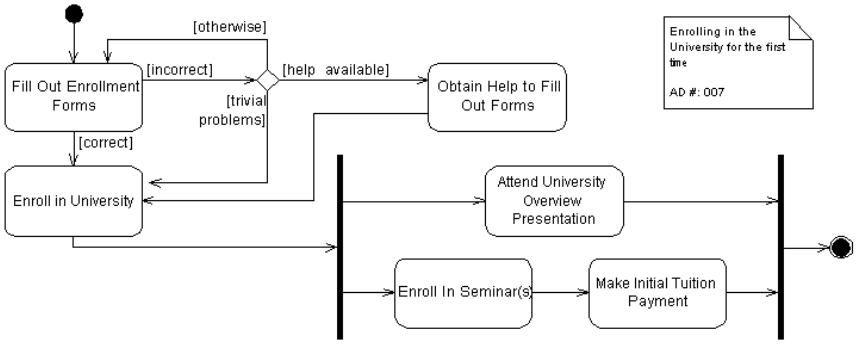


Diagram aktivnosti



Ne spada več pod mrežno igro – očitno!

Dokumentiranje kode

- Koda je dokumentacija!!! (XP)

- Struktura
- Preglednost; razdrobljenost na dele
- Berljivost kode
- Način poimenovanja spremenljivk,...
- Komentarji na vseh nivojih
- Izoliranost; združevanje sorodnih funkcij

XP: The design documentation is maintained by the engineers within the code. Indeed, the code itself *is* the design documentation!

•Uporaba poudarjenih komentarjev (uokvirjanje, podčrtavanje), kjer je to pomembno!
•Velikost posameznega komentarja, funkcije ali vsaj strukture (npr. vgnezdenih struktur) naj ne presega velikosti zaslona.
•Uporaba daljših imen odpravi potrebo po tipkanju večjega števila dodatnih komentarjev. Programi so hkrati tudi bolj berljivi.

•Izolacija detajlov in zmanjšanje medsebojne soodvisnosti funkcij (procedur, modulov,...) oziroma delov programa združenih v različnih datotekah.
•Število globalnih spremenljivk je potrebno kolikor je mogoče omejiti.

Različni programerji imajo različne stile pisanja programov. Najpomembneje je, da je program dobro berljiv. Vseeno pa je pri združevanju programov večih programerjev ugodneje, če se vsi držijo nekih osnovnih pravil.
Primer dobrega stila pisanja programov (ob že omenjenih napotkih):
•v eni vrstici naj bo samo en programski stavek,
•z uporabo praznih vrstic povečamo preglednost, vendar jih ne sme biti preveč, ker potem vidimo na zaslonu manjši kos programa,
•posamezna imena in operatorje ločimo s presledki tako, da se poveča preglednost,
•celotna koda naj ima enako obliko (presledki, oblika komentarjev,...),
•struktura se mora videti iz oblike teksta (zamikanje stavkov, ki so znotraj kontrolne strukture za tri stolpce oziroma en tabulator zamaknjeni v desno),
•vse spremenljivke morajo biti inicializirane pred uporabo,
•goto stavki motijo kontinuiteto programa in se jim moramo čimbolj izogibati – njihovo uporabo je potrebno omejiti samo na procesiranje napak in izjemnih stanj (skok iz večkrat vgnezdenih zank ipd.),
•vsako funkcijo pred implementiranjem najprej opišimo v naravnem jeziku in/ali psevdo kodo, ki razkrije tudi osnovno strukturo funkcije, kar nam služi tudi kot komentar funkcije,

•dolžina ene funkcije praviloma naj ne bo daljša od 50 vrstic (brez glave funkcije),
•logični izrazi (if, while,…) naj bodo čimbolj razumljivi – če niso, jih je potrebno preoblikovati (presledki, oklepaji,...) ali uporabiti makroje #define,
•najbolje je uporabiti algoritme in metode, ki so nam že znane, sicer pa je potrebno posebno dobro preveriti razne robne pogoje,
•v datotekah, ki jih vključujemo v druge datoteke s komando prevajalnika #include "...", naj bodo samo deklaracije funkcij, spremenljivk in podatkovnih struktur ter makroji. Datoteke, v katerih so programske funkcije, naj se prevajajo posebej in potem poveže (linka) v izvajalni program,
•trivialnih opravil ne dajemo v posebne funkcije, kaj šele v posebne datoteke, ker se s tem izgublja preglednost,
•kodiranja se lotimo šele, ko je problem že dovolj dobro obdelan in specificiran.

Vsebina glave funkcije

- datum in ime avtorja, ki je kreiral funkcijo,
- datum, avtorja in namen modifikacij,
- opis delovanja funkcije,
- opis uporabljenih algoritmov,
- posebnosti pri izvedbi (če so) – še posebej tiste, ki lahko povzročijo probleme pri spreminjanju kode,
- seznam uporabljenih funkcij (samo funkcij, ki niso del standardnih knjižnic),
- opis vhodnih parametrov funkcije,
- opis izhodne vrednosti funkcije,
- uporabljene globalne spremenljivke (če se jih uporablja)

Dodatno:

•številka/ime modula/datoteke, kjer se funkcija nahaja

•...

Podobno velja tudi za razlago podatkovnih struktur, razredov,...

Primer glave funkcije

```
/*****************************************************************************
*                                                                            *
*            V p i s _ p a r a m e t r o v _ v _ E E P R O M                 *
*                                                                            *
* Modul: 1.09    Vpis parametrov regulatorja v EEPROM.                       *
*                                                                            *
*============================================================================*
* Verzija Datum     Opis                                          Avtor      *
*----------------------------------------------------------------------------*
*  1  1992-04-02  Zacetna verzija                                B. Premzel *
*                                                                            *
*============================================================================*
*                                                                            *
* Sintaksa:                                                                  *
*    int Vpis_parametrov_v_EEPROM(struct eeprom_tab*);                       *
*                                                                            *
* Parametri: podatkovna struktura - tabela parametrov                        *
*           (podatkovna struktura je deklarirana v "vars.h")                 *
*                                                                            *
* Izhodna vrednost: 1 (TRUE) - pravilno vpisano                              *
*                   0 (FALSE) - napaka pri vpisovanju                        *
*                                                                            *
* OPIS:                                                                      *
*  Program vpise v EEPROM vse parametre, ki so podani v tabeli, ki je        *
*  v datoteki "eeprom.h".                                                    *
*  Napako javlja, ce je parametrov prevec ali ce pride do napake pri         *
*  vpisovanju v EEPROM (npr. time out pri poskusu vpisovanja v EEPROM -      *
*  napaka v EEPROM-u ali manjkajoc EEPROM.                                   *
*                                                                            *
*  Podatki so dodatno zasciteni s kontrolno vsoto, ki se jo racuna ob        *
*  shranjevanju in preverja pri nalaganju. Kontrolno vsoto se vpise v        *
*  EEPROM tik za parametri.                                                  *
*                                                                            *
*****************************************************************************/
```

Vsebina glave datoteke

- ime datoteke in kratek opis vsebine,
- podatki o avtorju, datumih in vsebini modifikacij,
- seznam vseh funkcij, ki so v tej datoteki,
- seznam glavnih podatkovnih struktur oz. glavnih skupin podatkovnih struktur,
- navodilo za prevajanje in povezovanje, če se le-to izvaja na neobičajen način (posebne opcije prevajalnika, povezovalnika,...),
- posebnosti pri uporabi funkcij te datoteke (npr. "Pred uporabo funkcij, ki delajo s podatkovnimi strukturami, je potrebno klicati funkcijo nalozi_podatke().")

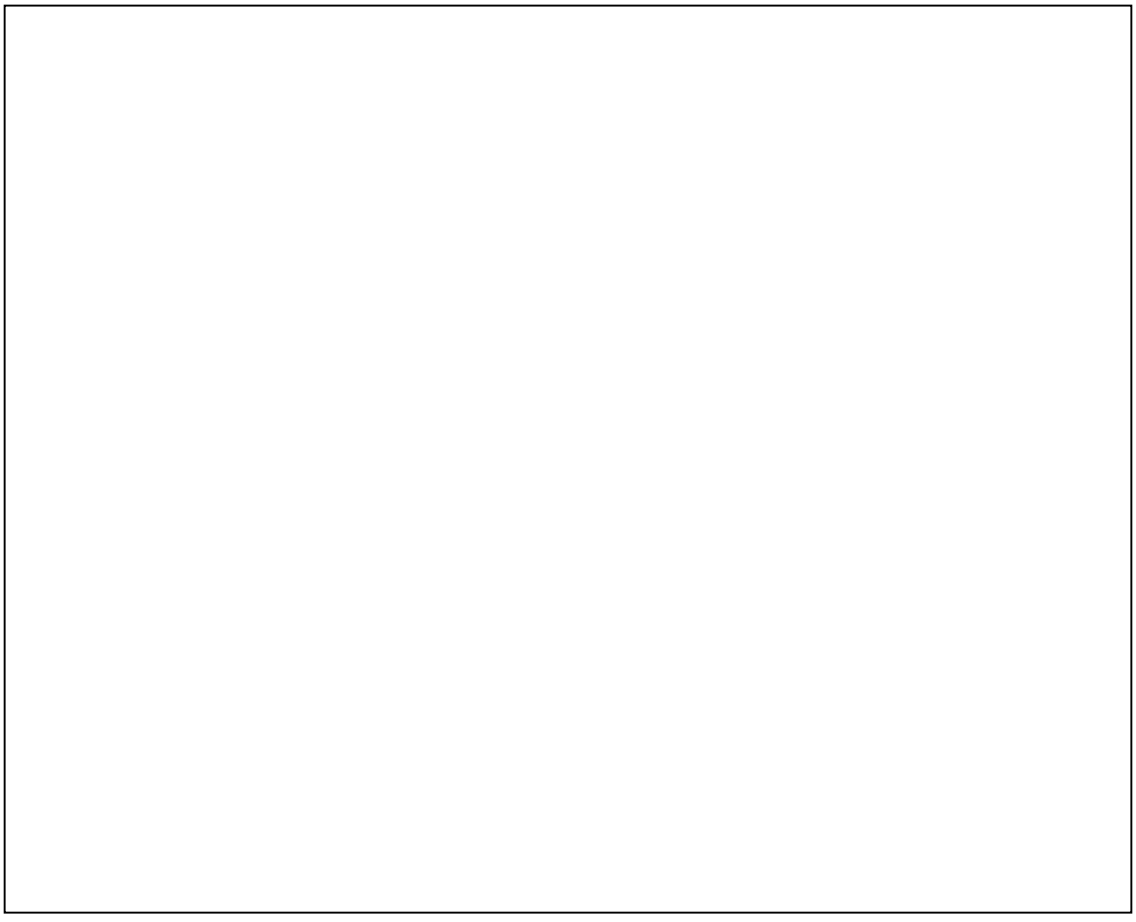
S primerno količino komentarjev se pridobi pri preglednosti, vendar ne na račun popolnosti dokumentacije.

Primer glave datoteke

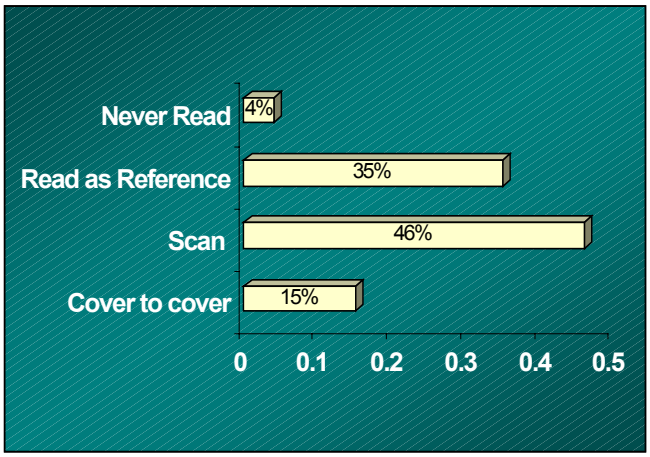
```
/*****************************************************************************
*                                                                            *
*                              E E P R O M . C                               *
*                                                                            *
* Citanje iz in pisanje v EEPROM, ki je na serijskem vodilu (I2C).           *
*                                                                            *
*============================================================================*
* Verzija Datum     Opis                                          Avtor      *
*----------------------------------------------------------------------------*
*  1  1992-04-10  Zacetna verzija                                B. Premzel *
*  2  1992-12-03  Predelava za novo verzijo regulatorja         B. Premzel *
*                                                                            *
*============================================================================*
*                                                                            *
* OPIS:                                                                      *
* Tukaj so zdruzene vse funkcije, ki shranjujejo informacije v EEPROM ali    *
* jih nalagajo iz njega:                                                     *
*                                                                            *
* void Brisanje_EEPROMa (int kaj);                                           *
*      Brisanje celotnega EEPROM-a (kaj = 1) ali samo shranjenih napak (0).  *
*                                                                            *
* void Zabelezi_cas_delovanja (void);                                        *
*      Pristeje cas delovanja regulatorja v vsoto, ki je v EEPROM-u.         *
*                                                                            *
* void Save_error (int vrsta_napake, unsigned int dodatna_inf_o_napaki);     *
*      Shrani informacije o napaki, ki je povzrocila izklop.                 *
*                                                                            *
* Funkcije, ki delajo direktno z aparaturno opremo, so v datoteki "i2c.a"    *
* in so realizirane v zbirnem jeziku (komunikacija z EEPROM-om).             *
*                                                                            *
*****************************************************************************/
```

Uporabniška dokumentacija

- navodila za namestitev
- konfiguracijski podatki
- uporabniški **priročnik**



Kako uporabniki berejo priročnike?



Razvojni cikel nastajanja publikacij

Koračni model:

1. Načrtovanje
2. Specifikacija vsebine
3. Pisanje
4. Produkcija
5. Ovrednotenje

Documentation can be considered as a project in project – podobnost z modeli vodenja projektov,...

By dividing a life cycle into phases you give yourself the opportunity to review the activities!

Načrtovanje

- Zberi informacije
- Uredi informacije v osnutek
- Preveri usnutek (potrditev)

- Glavna naloga: načrt za **cel** dokument
- Torej, nikoli ne začni s pisanjem prvega odstavka!
- Določi osnovni namen publikacije (opis arhitekture, funkcionalnosti, aplikacije novega orodja,...)
- Kakšne informacije potrebuje bralec?
- Horizontalno načrtovanje

Step into the shoes of your readers and determine what type of information they might be looking for

•Product description (what is it, what's its function, what components does it have, ….)

•Instructions (what is the task, what tools do I need, where do I start, what do I do next, what can I do if I go wrong)

•Am I proposing a change of policy? (what are the problems with the ways things were made till now?, what do you propose to do, what should happen next?)

Turn your questions into answers (Horizontal Planning)

•Write down your key questions and write a brief answers to each question

•When you complete your pre-plan you can organize missing information, organize information within sections, decide of a logical sequence

Specifikacija vsebine

- Zberi informacije
- Spiši specifikacijo vsebine
- Preveri specifikacijo (potrditev)

Vsebina specifikacije:

- Namen dokumenta
- Ciljno občinstvo
- Funkcija dokumenta
- Povezava z drugimi dokumenti
- Struktura dokumenta
- Povzetek vsebine
- Glavne odvisnosti in tveganja

Document specification

•Informs other in the project team of your intention

•It gives the project team the possibility to catch missing points at early stage

•It gives the team early warning on the equipment and resources you need to complete your task

•**The specification can be used as a skeleton of your manual**

Pisanje

- **Določi format publikacije**
- Spiši, uredi in ilustriraj
- Preveri osnutek (Edit Draft)
- Ustvari indeks

Document **design**, ideally, should not take place after the important activity of deciding what to say! The purpose of document design is not to confuse users even more than they already are!

Produkcija

- Prevod in lokalizacija
- Pripravi končno različico:
  - Zadnji pregled (Final Edits)
  - Ustvari indeks in kazalo
- Sodelovanje s prodajalcem

Ovrednotenje

- Spremljanje odziva
- Arhiviranje elektronskega materiala
- Arhiviranje natisnjenega materiala

Osnovna načela

- Kdo je naše občinstvo?
  - Strokovnjaki: usmeri jih direktno na problem-rešitev oziroma na del dokumentacije
  - Začetniki: usmeri jih na lažje dele dokumentacije
  - Inženirji, prodajalci, stranke,...
- (1) Kaj želimo opisati & *(2) kaj pričakujejo bralci?*
  - (1) Katere informacije naj predstavim v dokumentu?
  - *(2) Moram prebrati celo knjigo?*
  - (1) Kako naj organiziram dokument, da bo logično strukturiran?
  - *(2) Kako hitro lahko najdem informacijo, ki jo iščem?*
  - (1) Kako naj predstavim informacije brez preveliko teksta?
  - *(2) Razumem sporočilo? Je jasno podano?*
  - (1) Kako predstaviti informacije na zadovoljiv način?
  - *(2) Ali je povezava med poglavji očitna?*

Ocene

- Nove strani: 2.5 strani/dan
- Spreminjanje: 3.5 strani/dan
- Nespremenjene strani: 20 strani/dan
- Online teme: 3.5 teme/dan
- Prvi spisan osnutek nastane po 60% vsega načrtovanega časa: 20% - raziskava , 40% pisanje

Predstavitev informacij

- Tekst čez celo stran je utrudljiv – ustrezna oprema dokumenta
- Ustvarjanje uravnoteženih strani
- Konsistentna oznaka rubrik (poglavij)
- Uporaba slik, tabel in grafov kjer je to možno

Oprema dokumenta:

•System of headings and subheadings distinguished by typesize

•Ample space above and below headings

•Progressive indention; numbering,..

•Space between paragraphs. It allow users to get the meaning of the information

•Bulleted lists

•Notes, warnings

•Highlighting device, **bold,** *Italics*

Stil

- Vsebina je kar sporočamo, stil pa kako jo sporočamo!
- Piši enostavno, neposredno in točno
- Stavki naj bodo kratki
- Izogibaj se humorja in žargona
- **Bodi konsistenten**
- Predvidevaj vprašanja bralcev
- Izogibaj se opredelitvi spola (...*njegov* nasvet...)
- Zapisano naj zveni naravno

Uporabnik pričakuje, da si ti strokovnjak, zato se izogibaj frazam kot so: “priporočljivo je”, “lahko”,...

Še nekaj napotkov za konec #1

- Razlika v naslovih na različnih stopnjah naj bo različna vsaj za faktor 1,4
- Poudarek (bold) je ekvivalent enemu nivoju više
- Izpostavi poglavja – npr. uporaba črt, krožnih segmentov
- Uporaba do 20% senčenja naslovov
- Uporaba različnih pisav za naslove in ostal tekst
- Ne uporabljaj le velikih črk v naslovih
- Na dveh najvišjih stopnjah začni na naslednji strani

Še nekaj napotkov za konec #2

- Naslov na najvišjem nivoju naj bo vedno na novi, desni, lihi strani
- Naslovi se uporabljajo za hitro iskanje po dokumentu – namen: označevanje stopenj in vsebine
- Izogibaj se okrajšavam
- Ton pisanja naj bo ustrežljiv, a ne ravnodušen; nekako uraden
- Akcije, rezultati in komentarji naj bodo ločeni
- Opiši postopek pred podajanjem detajlnih korakov
- Navodila naj bodo varna

Še nekaj napotkov za konec #3

- Uporabljaj slikovni material, kjer je to možno – slika lahko pove več kot 1000 besed, pa tudi človek si tako podano informacijo lažje zapomni
- Ko uporabljaš zaslonske slike, ne uporabljaj resničnih imen računalnikov, uporabniških imen,...
- Uporabljaj tabele za smiselno strukturiranje in predstavitev informacij

Še nekaj napotkov za konec #4

- Uporabljaj barve, saj: pospešujejo iskanje, učenje je lažje, pomagajo pri sprejemanju odločitev,...
- Ne uporabljaj več kot sedem barv
- Barve uporabljaj konsistentno
- Uporabljaj barvne asociacije iz narave (rdeča – kri – nevarnost)
- Uporabljene barve naj imajo različno svetlost (barvna slepota, čb tisk)

Natisnjena : online dokumentacija

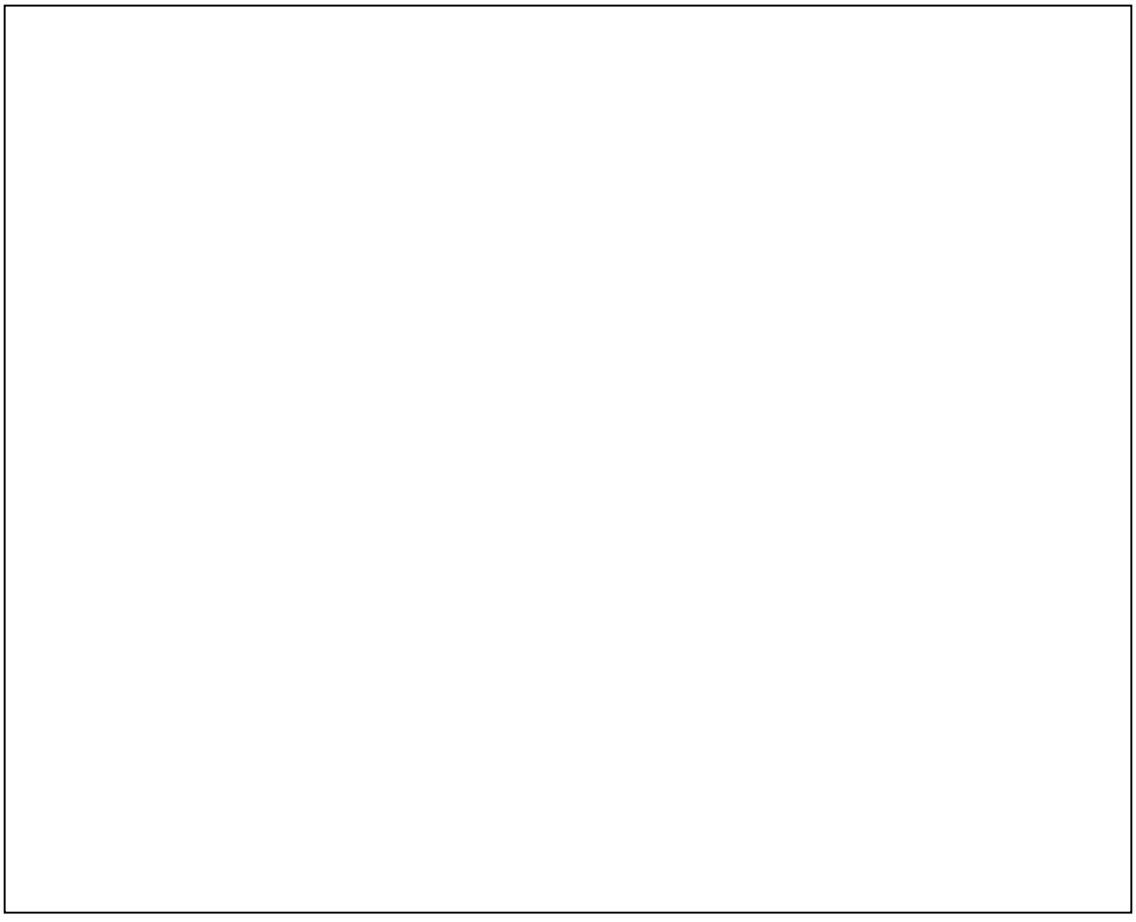
- Prednosti
- Slabosti
- Kdaj uporabiti kateri pristop?

Najboljša dokumentacija je dobro premišljena kombinacija natisnjenega in online materiala, kjer vsak medij izkorišča svoje pozitivne lastnosti!

Natisnjena dokumentacija #1

Prednosti:

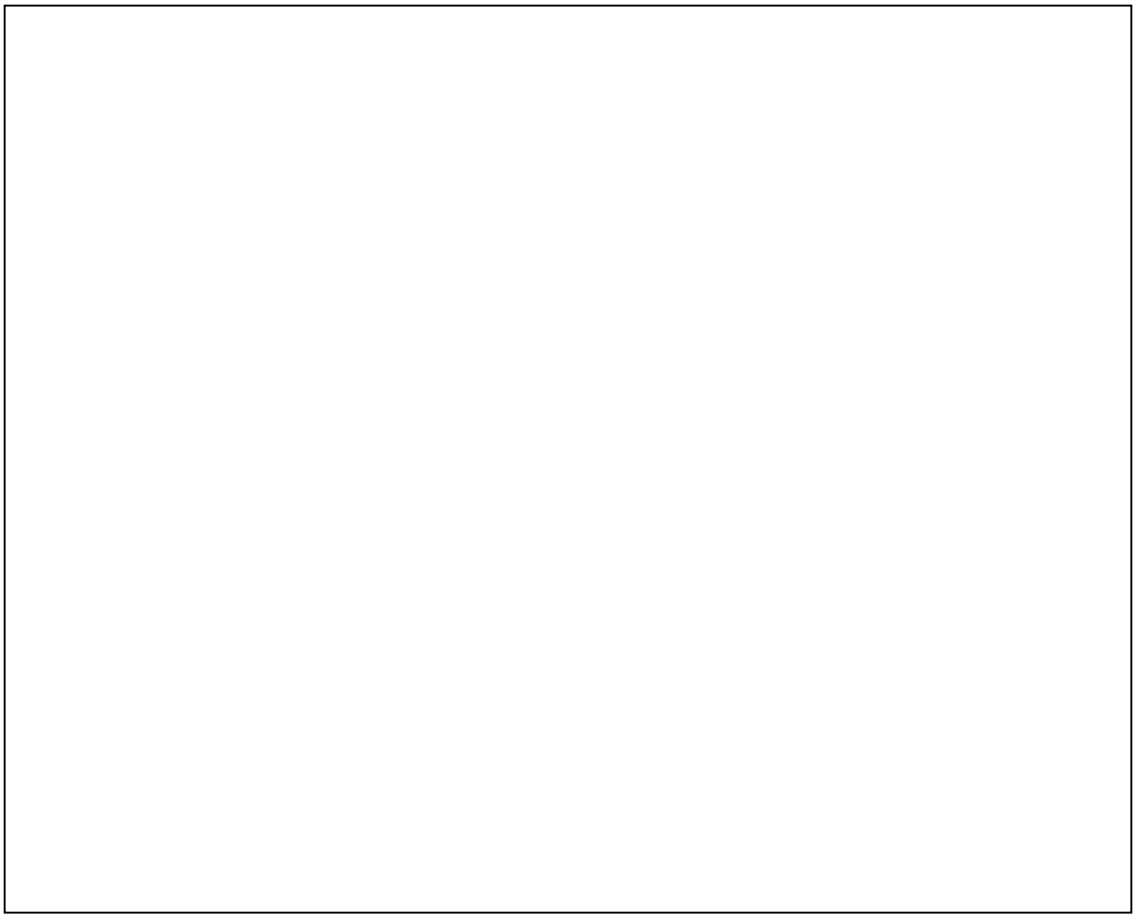
- Prenosljivost
- Večja ločljivost
- Lažje zapisovanje opomb, označevanje,...



Natisnjena dokumentacija #2

Slabosti:

- Največkrat nerodno in slabo formatirana
- Stroški produkcije so povezani s tiskom
- Nova različica zahteva zamenjavo
- Iskanje informacije zahteva fizično preiskovanje



Natisnjena dokumentacija #3

Kdaj uporabimo ta pristop?

- Uvajalni vodiči (Getting Started)
- Vodiči za odpravljanje napak (Troubleshooting)
- Material za hitri vpogled (Quick Reference)
- Materiali, ki jih rabimo, ko nismo ob računalniku
- Materiali za namestitev

Online dokumentacija #1

Prednosti:

- Dobro orodje za iskanje informacij
- Ni stroškov s tiskanjem
- Spremembe lahko enostavno integriramo

Online dokumentacija #2

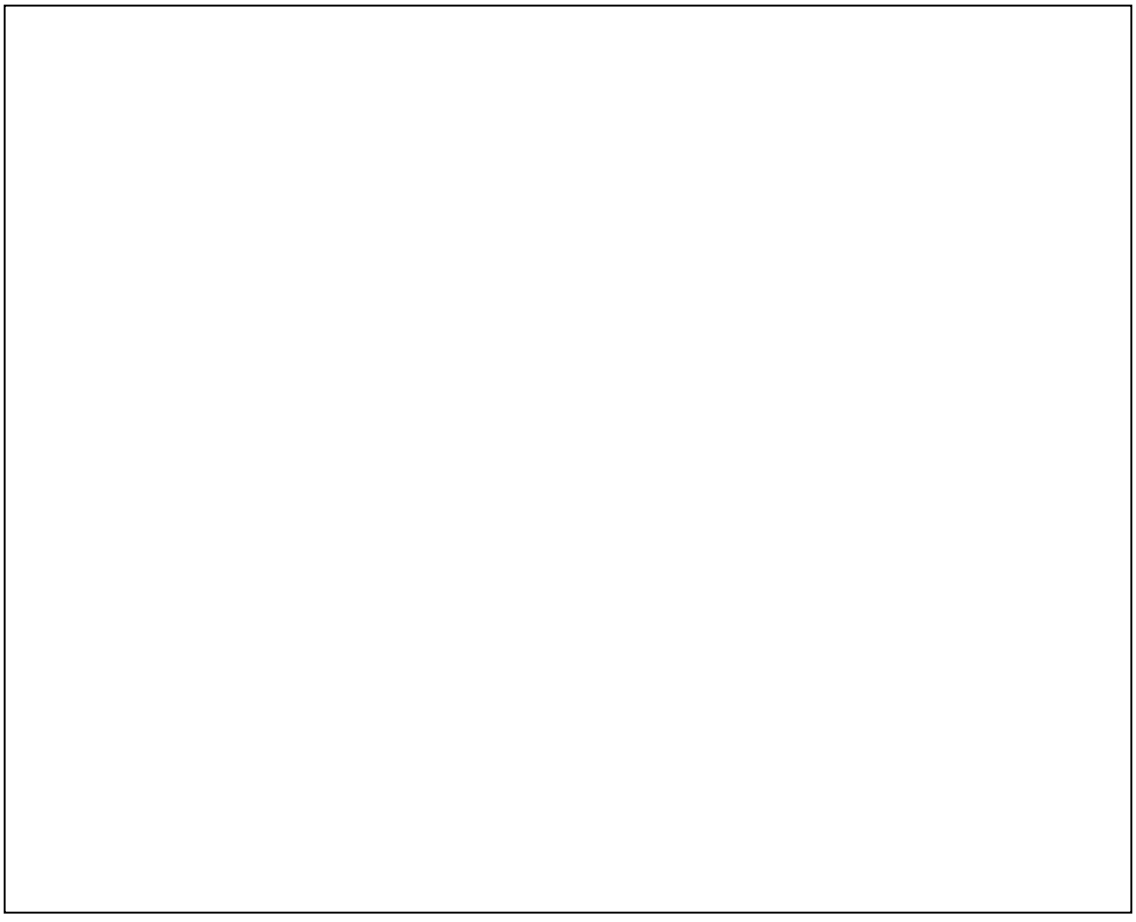
Slabosti:

- Zahteva računalnik, načeloma pa jo lahko tudi natisnemo
- Težja za uporabo za netehnične uporabnike
- Slabša ločljivost
- Razvojni čas je daljši
- Index in kazalo nista dosegljiva, če jo natisnemo

Online dokumentacija #3

Kdaj uporabimo ta pristop?

- Ko potrebujemo informacijo in delamo za računalnikom
- Za proceduralne naloge
- Za online odpravljanje napak



Sprotno dokumentiranje?

- Pojavljajo se primeri, kjer sprotno dokumentiranje za uporabnika pospeši celoten razvoj!
- XP + sprotno dokumentiranje za uporabnika ⇔ mogoče celo kot alternativa programiranju v parih?

Osnovna literatura

- M. Heričko: *Sodobni procesi razvoja*, prosojnice, FERI, llll?.
- A. Jaklič: *Agilne metodologije razvoja programske opreme*, predavanje, FRI, 2003.
- I. Jakša Zupančič: *User Documentation*, predavanje, HERMES SoftLab, 2001.
- M. Peternel: *Razvoj sodobne internetne računalniške igre*, seminarska naloga na podiplomskem študiju FRI, 2003.
- J. Pogorelc, M. Terbuc: *Specificiranje, dokumentiranje in izdelava programske opreme s programskim jezikom C*, zapiski predavanj, FERI, 2000.
- F. Solina: *Projektno vodenje razvoja programske opreme*, založba FE/FRI, 1997.
- L. Williams, A. Cockburn (gostujoča urednika): Agile Methods, *IEEE Computer*, junij 2003.
- Internet!

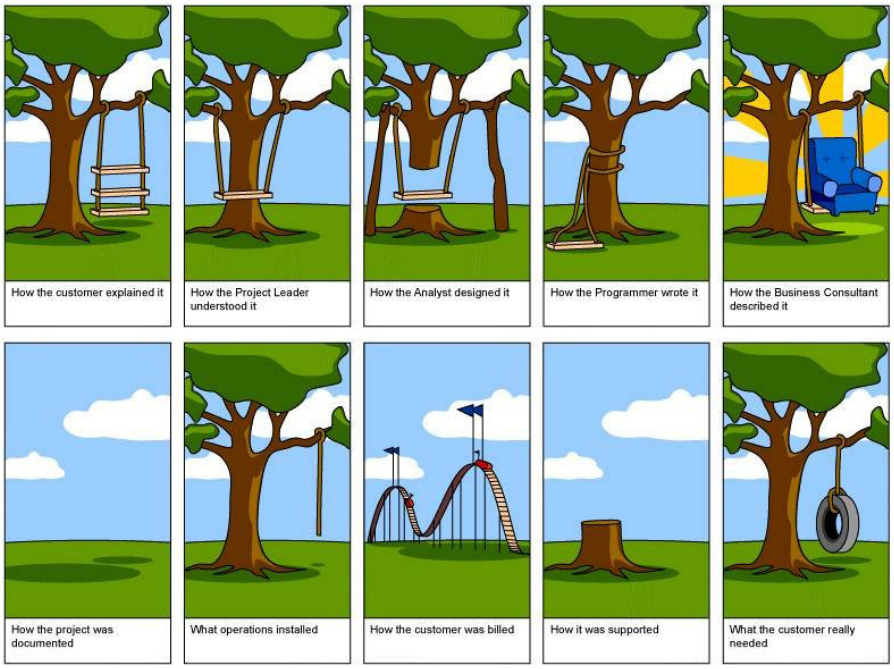
SSADM

UML

...

ISO

...


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed