DOKUMENTIRANJE

INFORMACIJSKE TEHNOLOGIJE

Peter Peer

Osnovne informacije

- DIT na spletu:
  http://www.lrv.fri.uni-lj.si/~peterp/DIT/DIT.htm
- Izpit je le pisni: 10 vprašanj!
- Tipična vprašanja???
  http://??? (Le kam je šel forum z VSŠ strani?)

Kaj je dokumentiranje?

DOKUMENTACIJA

≡

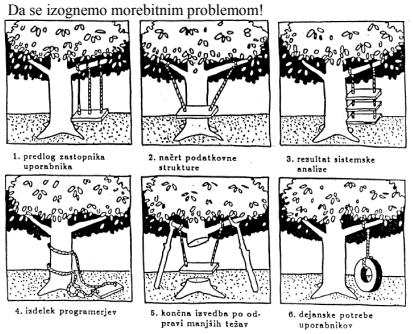
SLEDLJIVOST

≡

STANDARD

Zakaj je to sploh pomembno?

Da se izognemo morebitnim problemom!



Kakšni so produkti dokumentiranja?

Dokumentacijo delimo na dva sklopa:

- UPORABNIŠKA DOKUMENTACIJA
- RAZVOJNA DOKUMENTACIJA

Pa kaj je to standard?

Standard je
dokumentiran dogovor, ki vsebuje tehnične specifikacije ali druge natančne zahteve, ki naj bodo stalno uporabljane kot pravila oziroma **smernice.**

UPORABNIŠKA DOKUMENTACIJA

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

- tehnična poročila (tudi seminarske in **diplomske naloge**)
- članki
- **predstavitve**

Ali je struktura diplomske naloge pomembna?

SEVEDA JE! Saj niste dvomili, ne?

- Vsak dokument moramo oblikovati glede na njegov namen in funkcijo!!!
- Ena najbolj pogostih začetniških napak pri oblikovanju besedil je, da v enem dokumentu uporabljamo preveč različnih vrst pisav. – Ne moreš verjeti...

Obstaja standardna struktura seminarskih in diplomskih nalog?

1. **Naslov** dela, imena avtorjev, institucija in njen naslov, naslovi elektronske pošte avtorjev, datum in kraj nastanka dela.
2. **Povzetek**, v katerem v sto do dvesto besedah opišemo vsebino celotnega dela (v slovenščini in tujem jeziku).
3. **Uvod**, s katerim uvedemo bralca v delo in podamo pregled celotnega dela.
4. **Pregled** področja, s katerim se delo ukvarja, in sorodnih rešitev, če predlagamo v delu neko novo rešitev. Če gre za kratko besedilo, sta uvod in pregled lahko združena.

5. **Glavni del**, ki je običajno sestavljen iz večih poglavij (npr. zajem podatkov, uporabljene metode in tehnologija, opis lastnih rešitev, rezultati).
6. **Zaključek**, kjer povzamemo glavne rezultate naloge.
7. **Zahvala**, v kateri navedemo imena vseh, ki so nam pri delu pomagali, oziroma so delo omogočili. To so lahko fizične osebe ali tudi institucije, ki so financirale naše delo.
8. **Seznam** virov oziroma literature, na katero se sklicujemo v svojem delu.
9. **Dodatki** (npr. algoritmi, daljše matematične izpeljave itd.).

Osnovo standardne strukture torej lahko povzamemo kot: IMRAD

- **I**ntroduction (uvod)
- **M**ethod (metoda)
- **R**esult (rezultati)
- **A**nd
- **D**iscussion (diskusija)

(Primeri!!! (diploma, članek, navodila,...))

Kaj potrebujemo za ustvarjanje?

Seveda, najprej urejevalnik besedil!

Načini urejanja besedil:

- Vizualno urejanje (WYSIWYG)
- Logično urejanje !!!

Kaj je to logično urejanje?

- Ima **sintakso**, torej je sila podobno programiranju!!!
- Torej, osnovno besedilo moramo opremiti z ustreznimi ukazi.
- Zakaj se imenuje logični? Ker od uporabnika zahteva predvsem, da v besedilu označi njegove logične komponente.
- Omogoča, da se nam med samim ustvarjanjem besedila **ni potrebno ukvarjati z vizualnim izgledom** besedila, saj vizualni izgled posameznih logičnih enot besedila določi prevajalnik na enoten način s pomočjo posebnih slogovnih datotek.
- Primeri???

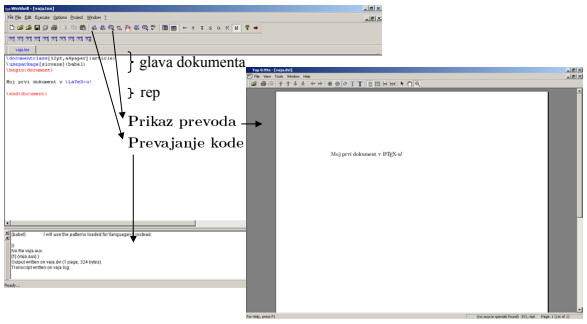
LaTeX

- Zelo primeren za pisanje **tehničnih** besedil!
- Standard na številnih založniških področjih.
- Uporablja se za tehnične in naravoslovne knjige, konferenčne zbornike, revije, slovarje in leksikone, večjezične knjige itd.
- Je v javni rabi in zato obstaja cela vrsta brezplačnih implementacij.
- MiKTeX: http://www.miktex.org
- WinShell: http://www.winshell.de

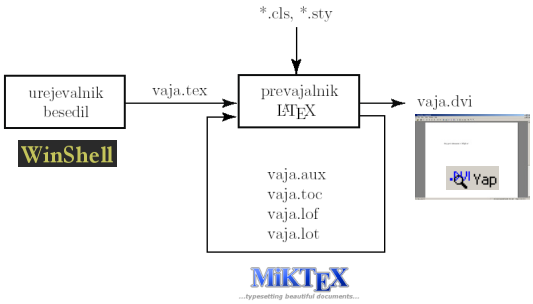
Literatura?

- M. Artač, B. Batagelj, M. Jogan, Ž. Kranjec, B. Kverh, K. Mele, **P. Peer**, M. Peternel, F. Solina: ***Uporabniška programska oprema***. 3. izdaja. Fakulteta za računalništvo in informatiko, Ljubljana, 2004. + Živi Linux: Slix
  http://www.lrv.fri.uni-lj.si/~franc/UPO04book/UPO04.html
- Za potrebe DIT je poglavje o LaTeX dostopno na spletni strani predmeta – ZGOLJ ZA INTERNO UPORABO!!!
  http://www.lrv.fri.uni-lj.si/~peterp/DIT/latex.pdf
- Na spletni strani knjige UPO je dostopen tudi primer uporabe!!!

WinShell grafični uporabniški vmesnik za MiKTEX ...typesetting beautiful documents...



Ozadje načina dela s sistemom in osnovne datoteke



Kakšna je vloga datotek?

- ***.tex** vhodna datoteka z besedilom in ukazi za formatiranje,
- ***.dvi** formatirana izhodna datoteka oziroma prevod datoteke *.tex,

- ***.aux** pomožna datoteka,
- ***.toc** kazalo,
- ***.lof** seznam slik,
- ***.lot** seznam tabel,
- ***.sty** stilske datoteke,
- ***.cls** oblikovne predloge.

Zgradba datotek s končnico **tex**?

```
\documentclass[12pt,a4paper]{article}
\usepackage[slovene]{babel}
\begin{document}

Moj prvi dokument v \LaTeX-u!

\end{document}
```

GLAVA

Vsebina

REP

⇓ prevod

Moj prvi dokument v LaTeX-u!

Komentar osnovnih lastnosti sistema?

- Glava in rep sta NUJNA!!!
- Vsak ukaz se začne z znakom \ !!!
- **12pt** podaja v pikah osnovno velikost pisave
- **a4paper** podaja velikost papirja
- **article** podaja vrsto dokumenta – **oblikovni vzorec**
- Paket **babel** (Babilon) - vse pomožne besede, ki se samodejno generirajo v dokumentu, se izpišejo v slovenščini

- Osnovni gradniki so okolja, ki imajo skupno osnovno sintakso:

  \begin{xyz**}**

  \end{xyz**}**
- Okolje vseh okolij:

  \begin{document}

  \end{document}

Kako pišemo šumnike?

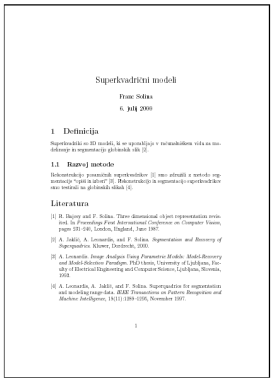
- Č – **"C**, š – **"s**,...
- Paket **latin2**:
  \usepackage[latin2]{inputenc}
  omogoča direkten vpis šumnikov

Slogi pisav?

\textrm pokončno - \textrm{pokon\v cno}
\textit *kurziva* - \textit{kurziva}
\textbf **krepko** - \textbf{krepko}
\textsc MALE KAPITELKE - \textsc{male Kapitelke}
\textsf neserifna pisava - \textsf{neserifna pisava}
\texttt pisalni stroj - \texttt{pisalni stroj}

Velikosti pisav?

\tiny beseda - {\tiny beseda}
\scriptsize beseda - {\scriptsize beseda}
\footnotesize beseda - {\footnotesize beseda}
\small beseda - {\small beseda}
\normalsize beseda - {\normalsize beseda}
\large beseda - {\large beseda}
\Large beseda - {\Large beseda}
\LARGE beseda - {\LARGE beseda}
\huge beseda - {\huge beseda}
\Huge beseda - {\Huge beseda}



Superkvadrični modeli

Franc Solina

6. julij 2000

1 Definicija

Superkvadriki so 3D modeli, ki se uporabljajo v računalniškem vidu za modeliranje in segmentacijo globinskih slik [2].

1.1 Razvoj metode

Rekonstrukcijo posameznih superkvadrikov [1] smo izvedli z metodo regresnega "spin in silver" [3]. Rekonstrukcijo in segmentacijo superkvadrikov smo testirali na globinskih slikah [4].

Literatura

[1] R. Bajcsy and F. Solina. *Three dimensional object representation revisited.* In *Proceedings First International Conference on Computer Vision*, pages 231–240, London, England, June 1987.

[2] A. Jaklič, A. Leonardis, and F. Solina. *Segmentation and Recovery of Superquadrics*. Kluwer, Dordrecht, 2000.

[3] A. Leonardis. *Image Analysis Using Parametric Models. Model-Recovery and Model-Selection Paradigm*. PhD thesis, University of Ljubljana, Faculty of Electrical Engineering and Computer Science, Ljubljana, Slovenia, 1993.

[4] A. Leonardis, A. Jaklič, and F. Solina. Superquadrics for segmentation and modeling range data. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 19(11):1289–1295, November 1997.

1

```
\documentclass[12pt,a4paper]{article} %%Ta dokument bo enokolonski!
\usepackage[slovene]{babel}

\begin{document}

\title{Superkvadriki\v cni modeli}
\author{Franc Solina}
\date{6. julij 2000}
\maketitle

\section{Definicija}
Superkvadriki so 3D modeli, ki se uporabljajo v ra\v cunalni\v skem
vidu za mo\v deliranje in segmentacijo globinskih slik
\cite{Kluwer_2000}.

\subsection{Razvoj metode}
Rekonstrukcijo posameznih superkvadrikov \cite{london_87} smo
zdru\v zili z metodo segmentacije ''spin\v si in iskari''
\cite{leonardis93}. Rekonstrukcijo in segmentacije superkvadrikov
smo testirali na globinskih slikah \cite{PAMI97}.

\begin{thebibliography}{1}
\bibitem{london_87}
R. Bajcsy and F. Solina. Three dimensional object representation
revisited. In {\em Proceedings First International Conference on
Computer Vision}, pages 231--240, London, England, June 1987.
\bibitem{Kluwer_2000}
A. Jakli\v{c}, A. Leonardis, and F. Solina. {\em Segmentation and
Recovery of Superquadrics}. Kluwer, Dordrecht, 2000.
\bibitem{leonardis93}
A. Leonardis. {\em Image Analysis Using Parametric Models:
Model-Recovery and Model-Selection Paradigm}. PhD thesis,
University of Ljubljana, Faculty of Electrical Engineering and
Computer Science, Ljubljana, Slovenia, 1993.
\bibitem{PAMI97}
A. Leonardis, A. Jakli\v{c}, and F. Solina. Superquadrics for
segmentation and modeling range data. {\em IEEE Transactions on
Pattern Recognition and Machine Intelligence}, 19(11):1289--1295,
November 1997.
\end{thebibliography}

\end{document}
```

Prevod in koda

Dvokolonsko?

Superkvadrični modeli

Franc Solina

6. julij 2000

1 Definicija

Superkvadriki so 3D modeli, ki se uporabljajo v računalniškem vidu za modeliranje in segmentacijo globinskih slik [2].

1.1 Razvoj metode

Rekonstrukcijo posameznih superkvadrikov [1] smo izvedli z metodo regresnega "spin in silver" [3]. Rekonstrukcijo in segmentacijo superkvadrikov smo testirali na globinskih slikah [4].

Literatura

[1] R. Bajcsy and F. Solina. Three dimensional object representation revisited. In *Proceedings First International Conference on Computer Vision*, pages 231–240, London, England, June 1987.

[2] A. Jaklič, A. Leonardis, and F. Solina. *Segmentation and Recovery of Superquadrics*. Kluwer, Dordrecht, 2000.

[3] A. Leonardis. *Image Analysis Using Parametric Models. Model-Recovery and Model-Selection Paradigm*. PhD thesis, University of Ljubljana, Faculty of Electrical Engineering and Computer Science, Ljubljana, Slovenia, 1993.

[4] A. Leonardis, A. Jaklič, and F. Solina. Superquadrics for segmentation and modeling range data. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 19(11):1289–1295, November 1997.

1

```
\documentclass[12pt]{article}
```



```
\documentclass[12pt,twocolumn]{article}
```

Za pravilno številčenje moramo dokument prevesti 2×!!!

Samodejno ustvarjanje kazal ter spiskov?

`\tableofcontents`	ustvari kazalo
`\listoffigures`	ustvari spisek slik
`\listoftables`	ustvari spisek tabel

Prevedemo 2× in dobimo rezultat!

Ostala osnovna okolja?

- Sredinska poravnava:
  \begin{center}
  \end{center}
- Naštevanje:
  \begin{itemize}
  \item
  \end{itemize}

- Številčenje:
  \begin{enumerate}
  \item
  \end{enumerate}
- Opisno naštevanje:
  \begin{description}
  \item[]
  \end{description}

- Okolje za dobesedni izpis:
  \begin{verbatim}
  \end{verbatim}
- Vrstični dobesedni izpis:
  \verb++
- Pisanje opomb:
  \footnote{}

- Okolje za poravnavo:

```
\begin{tabular}{|clr|}\hline
ena & dva & tri\\\cline{2-2}
1 & 2 & 3\\\hline
\end{tabular}
```

ena	dva	tri
1	2	3

- Okolje za poravnavo – združevanje stolpcev:

```
\begin{tabular}{|clr|}\hline
\multicolumn{2}{||c||}{1+2} & 3\\\hline
ena & dva & tri\\\cline{2-2}
1 & 2 & 3\\\hline
\end{tabular}
```

1+2		3
ena	dva	tri
1	2	3

- Tabela (plavajoče okolje):

```
\begin{table}[htb]
\begin{tabular}
...
\end{tabular}
\caption{}
\label{}
\end{table}
```

Sklic

```
V tabeli~\ref{rezultati} lahko...
```

V tabeli 1 lahko...

- Slika (plavajoče okolje):

```
\begin{figure}[htb]
\psfig{figure=,width=}
\caption{}
\label{}
\end{figure}
```

V glavo dokumenta moramo dodati:

```
\usepackage{graphics,epsfig}
```

Matematična okolja?

Poznamo dva tipa matematičnih okolij:

- vrstičnega

```
Komutativnost: $a+b=b+a$
```

Komutativnost: $a+b=b+a$

- sredinjenega

```
$$a+b=b+a$$
```

$$a+b=b+a$$

Okolje array:

```
$$
\left\{
{\bf X}=
\left[
\begin{array}{clr}
a+b & a-b & a+b+c\\
1 & 2 & 3
\end{array}
\right]+...\right.
$$
```

$$\left\{\mathbf{X}=\left[\begin{array}{clr} a+b & a-b & a+b+c \\ 1 & 2 & 3 \end{array}\right]+...\right.$$

`\left(`	veliki okrogli oklepaj
`\right)`	veliki okrogli zaklepaj
`\left[`	veliki oglati oklepaj
`\right]`	veliki oglati zaklepaj
`\left\{`	veliki zaviti oklepaj
`\right\}`	veliki zaviti zaklepaj
`\left|`	velika navpična črta na levi strani
`\right|`	velika navpična črta na desni strani
`\left.`	na levi strani ni prikazan oklepaj
`\right.`	na desni strani ni prikazan zaklepaj

- Samodejno številčenje enačb?

```
\begin{equation}
a+b=b+a
\label{Komutativnost}
\end{equation}
```

$$a + b = b + a \tag{5.1}$$

```
\begin{eqnarray}
a+b & = & b+a
\nonumber\\
a+(b+c) & = & (a+b)+c
\label{Asociativnost}\\
a*(b+c) & = & a*b+a*c
\label{Distributivnost}
\end{eqnarray}
```

$$\begin{aligned} a + b &= b + a \\ a + (b + c) &= (a + b) + c && (5.2) \\ a * (b + c) &= a * b + a * c && (5.3) \end{aligned}$$

Najpomembnejši ukazi za ustvarjanje matematičnih izrazov?

- Ulomki:

```
$$
x=\frac{a+b}{a-b}
$$
```

$$x=\frac{a+b}{a-b}$$

- Indeksi in potence:

```
$$
x_{22}=a^2+b^2
$$
```

$$x_{22}=a^2+b^2$$

- Integrali in odvodi:

```
$$
y=\int_0^{33}x^2dx \;\;\;\;\; z=y^{\prime\prime}
$$
```

$$y=\int_0^{33}x^2dx \;\;\;\;\; z=y^{\prime\prime}$$

\sqrt{x}	$\sqrt{x}$	\dot{a}	$\dot{a}$
\underline{x}	$\underline{x}$	\ddot{a}	$\ddot{a}$
\overline{x+y}	$\overline{x+y}$	\vec{a}	$\vec{a}$

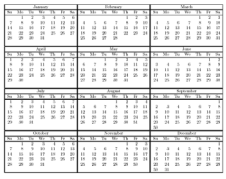
\leftarrow	$\leftarrow$	\Leftarrow	$\Leftarrow$
\rightarrow	$\rightarrow$	\Rightarrow	$\Rightarrow$
\leftrightarrow	$\leftrightarrow$	\Leftrightarrow	$\Leftrightarrow$

\oint	$\oint$	\nabla	∇	\sum	$\sum$	\forall	$\forall$
\prod	$\prod$	\exists	$\exists$	\bigcap	$\bigcap$	\neg	$\neg$
\bigcup	$\bigcup$	\le	$\le$	\bigvee	$\bigvee$	\ge	$\ge$
\bigwedge	$\bigwedge$	\subset	$\subset$	\pm	$\pm$	\subseteq	$\subseteq$
\cdot	$\cdot$	\equiv	$\equiv$	\times	$\times$	\propto	$\propto$
\Re	$\Re$	\in	$\in$	\Im	$\Im$	\notin	$\notin$
\partial	∂	\sim	$\sim$	\infty	∞	\simeq	$\simeq$
\emptyset	$\emptyset$	\approx	$\approx$	\neq	$\neq$	\doteq	$\doteq$

\alpha	α	\iota	ι	\varrho	ϱ
\beta	β	\kappa	κ	\sigma	σ
\gamma	γ	\lambda	λ	\varsigma	ς
\delta	δ	\mu	μ	\tau	τ
\epsilon	ϵ	\nu	ν	\upsilon	υ
\varepsilon	ε	\xi	ξ	\phi	ϕ
\zeta	ζ	\o	o	\varphi	φ
\eta	η	\pi	π	\chi	χ
\theta	θ	\varpi	ϖ	\psi	ψ
\vartheta	ϑ	\rho	ρ	\omega	ω
\Gamma	Γ	\Xi	Ξ	\Phi	Φ
\Delta	Δ	\Pi	Π	\Psi	Ψ
\Theta	Θ	\Sigma	Σ	\Omega	Ω
\Lambda	Λ	\Upsilon	Υ		

In še več?

- Seveda, omogoča tudi barvni tisk
- Pravi programski jezik, v katerem je možno celo računati



Prednosti vizualnega urejanja besedil?

- Orodja za vizualno urejanje besedil je lažje uporabljati in se jih uporabniki hitreje naučijo.
- Z vizualnimi orodji je lažje izvajati zahtevno grafično oblikovanje.
- Primerna so predvsem za kratka besedila, za dolga besedila pa kmalu postanejo preokorna.

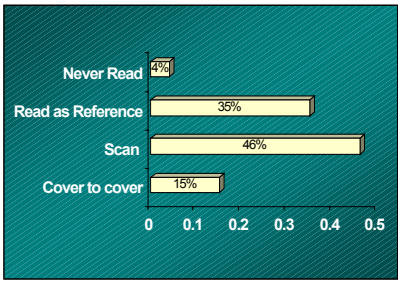
Prednosti logičnega urejanja besedil?

- Logično urejanje zaradi ločitve vsebine (logične strukture besedila), od oblike omogoča konsistentno oblikovanje celotnega besedila na osnovi njegove logične strukture.
- Logično strukturirana besedila lahko prevedemo iz ene strukturirane oblike v drugo strukturirano obliko (npr. LaTeX v HTML ali obratno), ali pa jih ustvarimo z drugimi računalniškimi orodji (npr. enačbe v formatu LaTeX s programom Mathematica).
- Dosežemo lahko veliko višjo in konsistentno tipografsko kvaliteto.
- Lažje prenosljive in veliko manjše datoteke (ASCII).

Primer fleksibilnosti logičnega urejanja besedil

- Fleksibilnost logičnega urejanja besedil omogoča ločitev strukture in oblike (stilske datoteke)!!! ⇒
- V izvornem besedilu v formatu LaTeX bomo pisali notranji produkt na naslednji način: **\np{A}{B}**.
- Z definicijo makro ukaza **\np** za notranji produkt:
  \newcommand[2]{\np}{(#1,#2)}
  notranji produkt A in B izpišemo kot (A,B).
- Le z ustrezno spremembo makro ukaza pa lahko notranji produkt **povsod** v besedilu izpišemo tudi na druge načine: (A, B), $(A|B)$ ali $A|B$.

Kako uporabniki berejo priročnike?



Koračni model nastajanja priročnika?

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

Kakšne informacije potrebuje bralec?

Uporabimo pristop

horizontalnega načrtovanja:

- Zapiši bistvena vprašanja, na katere naj bi odgovarjal priročnik, in podaj kratke odgovore na njih!
- Na podlagi takšnega osnutka lahko sedaj lažje določiš, kaj še manjka, reorganiziraš informacije znotraj poglavij, določiš logično zaporedje poglavij,...

Temeljna vprašanja bralca?

- Moram prebrati celo knjigo?
- Kako hitro lahko najdem informacijo, ki jo iščem?
- Razumem sporočilo? Je jasno podano?
- Ali je povezava med poglavji očitna?

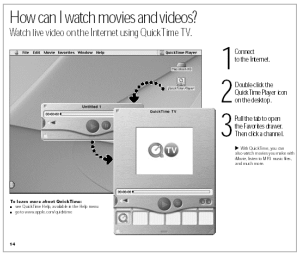
Vsebina specifikacije?

- **Namen** dokumenta
- Ciljno **občinstvo** (strokovnjaki, začetniki,...?)
- Funkcija dokumenta
- Povezava z drugimi dokumenti
- **Struktura** dokumenta
- **Povzetek** vsebine
- Glavne odvisnosti in tveganja

Kdaj določiti format publikacije?

Pred samim PISANJEM, saj

format vpliva na način podajanja vsebine!!!



Kako predstaviti informacije? (<u>format</u>, <u>vsebina</u>, <u>stil</u>)

format

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

stil

- Piši enostavno, neposredno in točno
- Stavki naj bodo kratki
- Zapisano naj zveni naravno
- **Bodi konsistenten**
- Predvidevaj vprašanja bralcev
- Izogibaj se humorja in žargona
- Izogibaj se opredelitvi spola (...*njegov* nasvet...)
- Izogibaj se frazam kot so: "priporočljivo je", "lahko",...

format

- 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

- 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

stil

- Izogibaj se okrajšavam
- Ton pisanja naj bo ustrežljiv, a ne ravnodušen; kot da ta trenutek razlagaš uporabo na štiri oči
- Akcije, rezultati in komentarji naj bodo ločeni
- Opiši postopek pred podajanjem detajlnih korakov
- Navodila naj bodo varna

- 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

- 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 ali 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

Prednosti:

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

Slabosti:

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

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

Prednosti:

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

Slabosti:

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

Kdaj uporabimo ta pristop?

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

Govorne predstavitve

- pogosto moramo predstaviti rezultate svojega dela (npr. **zagovor diplomske**)
- na predstavitev se moramo dobro pripraviti
- upoštevati moramo predvsem, **kdo** nas bo poslušal in koliko **časa** imamo na voljo za predstavitev
- za svoj nastop moramo pripraviti **ustrezna gradiva** in svoj nastop **vaditi**
- spontan nastop zahteva veliko vaje!

Gradiva za govorno predstavitev?

- gradiva za udeležence
- gradiva za govornika
- projekcijska gradiva

Kako pripraviti projekcijska gradiva?

- vsebinski **izbor** tega, kar nameravamo povedati
- ena **minuta** govora $\Rightarrow$ ena **prosojnica**
- na eno prosojnico: kolikor lahko napišemo s tiskanimi črkami na listek **8×8 cm**
- čimveč informacij podamo na **grafičen** način

- temna pisava na svetlem ozadju ali svetla pisava na temnem ozadju? ⇒ **kontrast!**
- ozadje naj bo **uniformno**!
- črke dovolj velike, da lahko berejo poslušalci iz zadnje vrste
- ne pretiravajmo z efekti (animacije, zvok, video), da ne zasenčijo govornika
- že pripravljene oblikovne predloge ali oblikovanje lastnih?

- MMG: **pdflatex** besedilo oblikovano v sistemu LaTeX pretvori v PDF, stilna datoteka **pdfslide.sty** pa omogoči dodatne ukaze za izdelavo interaktivne predstavitve

RAZVOJNA DOKUMENTACIJA

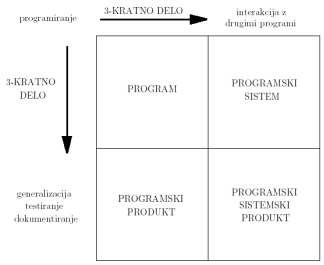
Pomen in nastajanje bomo spoznali preko:

- Projektnega dela
- Razvojnih modelov
- Programskega inženirstva

Kaj je to projekt?

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.**

Ali je programsko inženirstvo programiranje?



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

Kaj obsega programsko inženirstvo?

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

Katere so torej faze projekta?

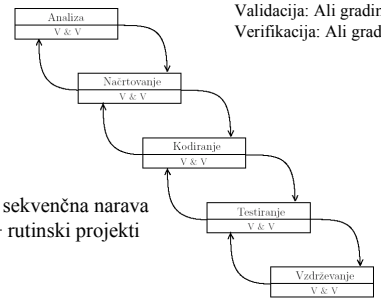
- Analiza
- Načrtovanje
- Implementacija
- Testiranje
- Vzdrževanje

Dokumentiranje

Osnovni razvojni modeli

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

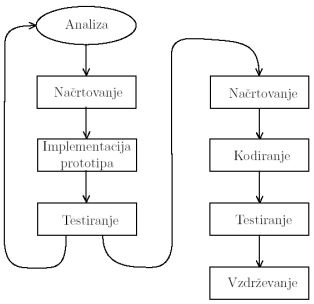
Klasični razvojni cikel



Validacija: Ali gradimo sistem pravilno?
Verifikacija: Ali gradimo pravi sistem?

- sekvenčna narava
+ rutinski projekti

Razvoj z uporabo prototipov



Kadar **zahteve** niso povsem jasno določene! (zajemanje zahtev)

Graditi iz prototipa končni sistem?
Hišica iz kart?

Razvoj s 4. generacijo programskih jezikov

- Gre predvsem za t.i. **programske generatorje**
- Poslovne aplikacije
- Baza podatkov v ozadju
- **Postopek** dela: načrtovanje logičnega, fizičnega modela baze, določitev oblike in lastnosti GUV, generiranje aplikacije, dodajanje funkcionalnosti (3. GPJ)

- **Prednost**: manjša potreba po znanju programiranja
- **Slabosti**: vzdrževanje mešane kode (3.+4. GPJ), časovni vložek je primerljiv

Primer!!!

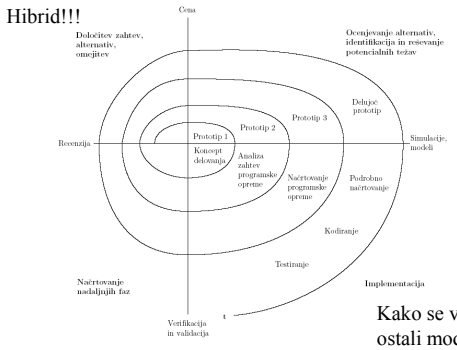
(Sybase Inc. – PowerBuilder/Designer (analiza po SSADM – planska metoda))

Postopni razvoj

Redna uporaba

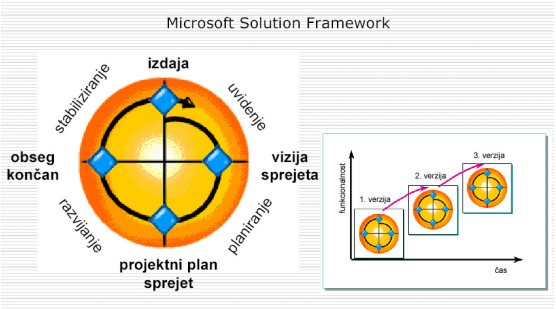
Spiralni razvoj

Hibrid!!!



Kako se v njem kažejo ostali modeli?

Primer iz prakse – MSF kot procesno ogrodje



(kaskadni pristop – **planska** metoda)

Planska ali agilna metoda?

	PLANSKA	**AGILNA**
Primarni cilji	napovedljivost, stabilnost, visoko jamstvo	hitro vidni rezultati, odzivnost na spremembe
Naročnik	interakcija po potrebi, osredotočenost na izvajanje pogodbe	kolociran, osredotočenost na majhne inkremente

	PLANSKA	AGILNA
Razvoj	obsežen načrt, dolgi inkrementi	enostaven načrt, kratki inkrementi
Kultura dela	udobje in moč skozi ogrodje načrtov in reda	udobje in moč skozi ponujeno svobodo pri delu

In katero bi izbrali vi? In zakaj?

Kaj pa pravi praksa?

- Problem (neuspeh) **planskih metod** je velika poraba časa in nepripravljenost na 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

"tipična" interakcija naročnik-razvijalec ☺

Torej agilna metoda?

Metoda je agilna, ko je razvoj programske opreme:

- **Inkrementalen** – majhne izdaje, hitri razvojni cikli ⇒ hitre povratne informacije
- **Kooperativen** – naročnik in razvijalci tesno sodelujejo
- **Neposreden** – metoda je enostavna, torej lahko naučljiva, prilagodljiva in dobro opisana
- **Prilagodljiv** – lahko obvlada spremembe zahtev

http://www.agilemanifesto.org

Cilj agilnosti?

Zmanjšanje časa, stroškov in povečanje fleksibilnost!
Bistven je delujoč program!

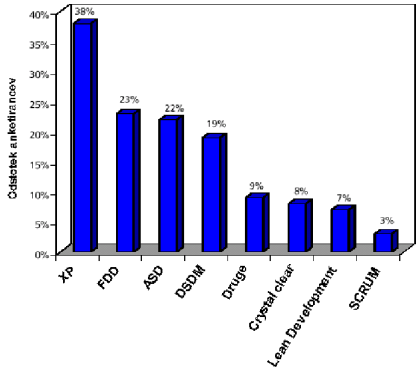
A **pozor**, agilnost v smislu:

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

in NE v smislu:

- hitrosti razvoja na račun kakovosti
- minimalnega opisa procesa, ker bi potem bilo najbolje kar brez procesa

Agilne metode?



Kako uvesti metodo v podjetje?

1. Poenotiti osnovni stil programiranja ZAKAJ?
2. Način arhiviranja
3. Samodejno generiranje razvojne dokumentacije

4. In ostale prakse?

Dokumentiranje kode?

- Koda je dokumentacija!!! (XP)
- **Berljivost** kode!
- Struktura
- Preglednost – razdrobljenost na dele
- Način poimenovanja spremenljivk,...
- Komentarji na vseh nivojih
- Izoliranost – združevanje sorodnih funkcij

- Uporaba **poudarjenih** komentarjev (uokvirjanje, podčrtavanje), kjer je to pomembno (pogodba)!
- **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.

- 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
- vsako funkcijo/metodo pred implementiranjem najprej opišemo v **naravnem** jeziku in/ali psevdo kodo, ki razkrije tudi osnovno strukturo funkcije, kar nam služi tudi kot komentar funkcije
- **kodiranja** se lotimo šele, ko je problem že dovolj dobro obdelan in specificiran
- **logični** izrazi (if, while,…) naj bodo čimbolj razumljivi – če niso, jih je potrebno preoblikovati (presledki, oklepaji,...)

Pogodba in podpis funkcije/metode/...?

- Podpis: `public Image getImage(URL url, String name)`
- Pogodba:

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 *         name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
```

Zakaj pogodba, zakaj podpis?

Kaj pogodba tipično vsebuje?

- Opis delovanja
- Parametrov
- Izhodne vrednosti
- Izpostavitev sorodnih konstruktov
- Avtor
- Datum
- Posebnosti

Primer!!!

Arhiviranje?

- **Koliko časa vam vzame (ob zaključku dneva):**
  - Arhiviranje delovnega direktorija (.zip)
  - Preimenovanje arhiva tako, da mu dodamo današnji datum
  - Ustvarimo tekstovno datoteko z enakim imenom, v katero zapišemo narejene spremembe (lahko že sproti)
  - Obe datoteki prenesemo na arhivski CD, kjer jih uredimo po datumu?
- **Kaj je <u>5 minut</u> v primerjavi z izgubljenim dnevom ali celo več, pri čemer imamo tudi popolnejšo razvojno dokumentacijo?**
- Seveda, s **CVS** naredimo še korak naprej!

Primer!!!

CVS?

- *Concurrent Versions System*
- Sistem za upravljanje različic

- **Oglejmo si osnovno delovanje!** (*checkout, commit, release, diff, log,...*)

- Arhiviramo le skladišče

Upravljanje različic s CVS (Concurrent Versions System)
Peter Peer, peter.peer@fri.uni-lj.si, za XYZ

Spletna stran:
http://www.cvshome.org/
Stabilna različica (9.5.2004):
http://ftp.cvshome.org/release/binaries/win32/cvs-1-11-15.zip
Priročnik:
http://ftp.cvshome.org/release/stable/cvs-1.11.15/cvsmanual-1.11.15.pdf

Ustvari skladišče:
cvs -d C:\Progra~1\CVS init
- v danem direktoriju se ustvari direktorij CVSROOT, ki vsebuje administracijske datoteke CVS
- ostali direktoriji, ki jih ustvarimo z ukazom spodaj, so v istem direktoriju kot CVSROOT in vsebujejo uporabniške module (dejanska skladišča)
- če kdajkoli ponovno zaženemo ta ukaz, se že obstoječe stvari ne prepišejo

Uvozimo projekt, ki ga želimo upravljati (postavimo se v direktorij, ki ga želimo upravljati):
cvs -d \localC:/Progra~1/CVS import -m "Uvoz datotek" SecurityAgentADO denali start
- oznaka *denali* označuje recimo naročnika
- oznaka *start* označuje izdajo
- pozor: uporabljeni so ?
- če opcije –m ni, se zažene urejevalnik, kamor lahko vpišemo željeni komentar ("Uvoz datotek"); komentar se vedno zapiše v vsako datoteko skladišča

Sedaj najprej arhivirajmo originalni direktorij in ga nato izbrišimo, da se ne bo slučajno kdaj zgodilo, da bomo popravljali vsebino originalnega direktorija, ki ni vezan na CVS!

Sedaj je skladišče vzpostavljeno in če želimo delati na določenem projektu, moramo najprej dobiti lastno kopijo projekta (postavimo se v direktorij, kamor želimo dati mi delovni direktorij):
cvs -d \localC:/Progra~1/CVS checkout SecurityAgentADO

Ko končamo s spreminjanjem moramo spremembe zapisati v skladišče (smo v našem delovnem direktoriju):
cvs commit my.ADO.h
- ker nismo uporabili opcije –m, se zažene urejevalnik, kamor vpišemo komentar revizije, npr.: Dodan optimizacijski preboj: optimizacija berljivosti

…

Ustvarjanje razvojne pomoči?

...z ala JavaDoc pristopom

- JavaDoc – orodje za ustvarjanje dokumentacije v HTML formatu iz komentarjev v izvorni kodi
- Sintaksa
- Razčlenjevalnik

- Rezultat: pomoč za razvoj (uporabo izvorne kode), del razvojne dokumentacije

- Ideja: razvoj lastnega, prirejenega, enostavnega generatorja **ali** uporaba JavaDoc ali podobnega pristopa

Primer izvorne kode

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 *         name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
    try {
        return getImage(new URL(url, name));
    } catch (MalformedURLException e) {
        return null;
    }
}
```

Ustvarjen HTML

getImage

```
public Image getImage(URL url,
                      String name)
```

Returns an `Image` object that can then be painted on the screen. The `url` argument must specify an absolute `URL`. The `name` argument is a specifier that is relative to the `url` argument.
This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:
`url` - an absolute URL giving the base location of the image
`name` - the location of the image, relative to the `url` argument

Returns:
the image at the specified URL

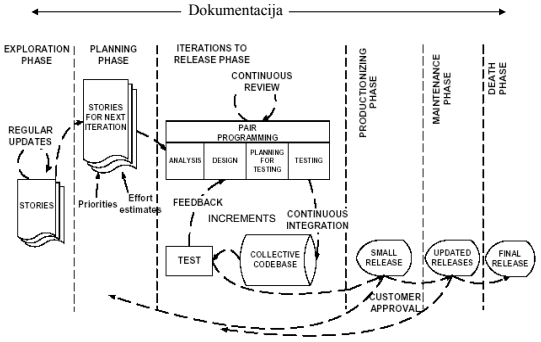
See Also:
`Image`

XP – eXtreme Programming

- Agilna metoda razvoja
- Zakaj ekstremna? Ker popelje dobre prakse do ekstremov!
- 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?



Uporabniške zgodbe?

- Uporabniški primeri, ki kratko opišejo **zahteve** za sistem v naravnem jeziku
- 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, planskih 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!
- So osnova za podajanje uspešnosti dela (**kazalec uspešnosti**)

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
- 4. spremenljivke: stroški, čas, obseg, **kakovost** – za slednjo odgovarjajo razvijalci
- Srečanja ob začetkih iteracij: izbira uporabniških zgodb za to iteracijo

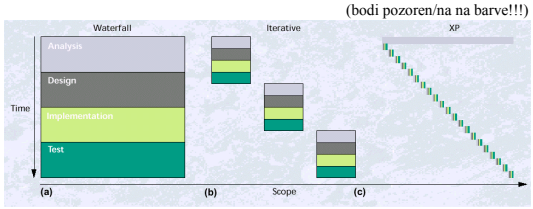
Igra planiranja?

- "XP-merji" planirajo
- Igra planiranja (up. zgodbe na karticah):
  cilji (plan izdaj različic),
  elementi igre (uporabniške zgodbe),
  igralci (naročnik, razvijalci) in
  poteze (premikanje kartic)

Iterativnost in inkrementalnost?

- Pri **inkrementalnem** pristopu razdelimo problem na **podprobleme** (**pod-uporabniške zgodbe**) tako, da jih lahko rešujemo sočasno in/ali izmenično. Ob rešitvi vsakega podproblema le-tega **testiramo** in **integriramo** z ostalimi deli sistema.
- **Iteracija** pa je en prehod skozi vse aktivnosti procesa.

(bodi pozoren/na na barve!!!)



od planskih k agilnim

Poudarek na skupinskem delu!

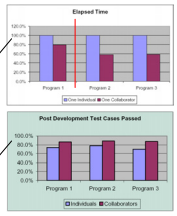
- Lastnik projekta je celotna skupina in vsak član lahko sodeluje na kateremkoli delu
- Zagovorniki XP trdijo, da je par več kot 2× produktivnejši kot posameznik
- Programiranje v parih (“pair programming”)

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**
- **Rotiranje** parov razvijalcev
- Programiranje v paru ni enako sedenju pred zaslonom in opazovanju nekoga, ki tipka!!!

http://www.pairprogramming.com

Ekonomičnost???



+15%

-15%

50.000 LOC (Lines Of Code)

Individualno delo	Sodelovanje	
1000 h	1150 h	← 50 LOC/h
1500 napak	1275 napak	← 100 napak/1000 LOC × 0,3 (za 70% lahkih napak)
225×10 = 2250 h	0 h	← 10 h/napako (znotraj testnega oddelka, sicer ×4)
3250 h	**1150 h**	

AND TIME IS **MONEY!!!**

Ostale prednosti?

- Zadovoljstvo

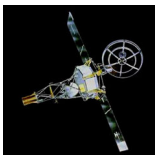
"The adjustment period from solo programming to collaborative programming was like eating a hot pepper. The first time you try it, you might not like it because you are not used to it. However, the more you eat it, the more you like it."

- Kvaliteta (krajši programi z manj napakami)
- Hitrejše reševanje problemov
- Učenje eden od drugega
- Povečana komunikativnost
- Večji pregled vsakega posameznika nad celotnim projektom (zmanjšana tveganost ob manjkanju člana)

Še vseeno potrebuješ alternativo?

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.



Pomembnost testov?

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`.

$$$

Testno voden razvoj?

- **Najprej napišemo teste na osnovi zgodb, nato šele kodo, ki zgodbe realizira**
- Testiranje mora potekati **avtomatsko**
- **Da napišemo teste, moramo razumeti zgodbe**
- Testi enote (Unit Tests) – definirajo in izvedejo jih razvijalci
- Funkcijski testi (Functional/Acceptance Tests) – definirajo jih naročnik, izvedejo razvijalci
- Avtomatsko testiranje kot osnova za **preoblikovanje** (refactoring)

Primeri???

Kako spišem in zaženem enostaven test v JUnit?

1. Create a subclass of `TestCase`.

```
package junitfaq;

import java.util.*;
import junit.framework.*;

public class SimpleTest extends TestCase {
```

2. Write a test method to assert expected results on the object under test.

```
    public void testEmptyCollection() {
        Collection collection = new ArrayList();
        assertTrue(collection.isEmpty());
    }
```

3. Write a `suite()` method that uses reflection to dynamically create a test suite containing all the `testXXX()` methods:

```
    public static Test suite() {
        return new TestSuite(SimpleTest.class);
    }
```

4. Write a `main()` method to conveniently run the test with the textual test runner:

```
    public static void main(String args[]) {
        junit.textui.TestRunner.run(suite());
    }
}
```

5. Run the test:

- To run the test with the textual test runner used in `main()`, type:

```
java junitfaq.SimpleTest
```

The passing test results in the following textual output:

```
.
Time: 0
OK (1 tests)
```

- To run the test with the graphical test runner, type:

```
java junit.swingui.TestRunner junitfaq.SimpleTest
```

The passing test results in a green bar displayed in the graphical UI.

When should tests be written?

Tests should be written before the code. Test-first programming is practiced by *only* writing new code when an automated test is failing.

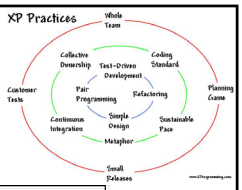
Good tests tell you how to best design the system for its intended use. They effectively communicate in an executable format how to use the software. They also prevent tendencies to over-build the system based on speculation. When all the tests pass, you know you're done!

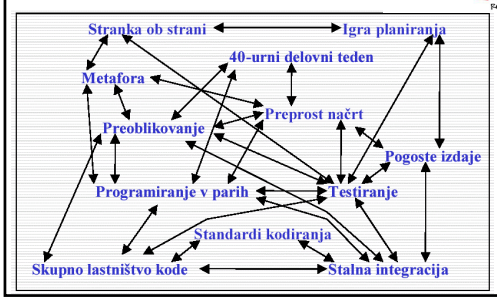
Whenever a customer test fails or a bug is reported, first write the necessary unit test(s) to expose the bug(s), *then* fix them. This makes it almost impossible for that particular bug to resurface later.

Test-driven development is a lot more fun than writing tests after the code seems to be working. Give it a try!

http://www.junit.org/

Prakse XP?





Metafora?

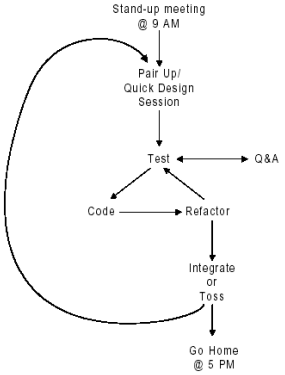
- Pojmi, ki se uporabljajo pri komunikaciji morajo imeti enak pomen za **vse** vpletene!!!
- Tipična napaka: razvijalec si razlaga določen pojem drugače kot naročnik, zato je treba **besednjak dobro definirati**!

Preoblikovanje kode?

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

Primer???

Tipičen dan programerja



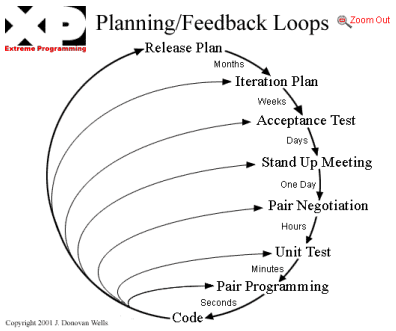
⇒ 40-urni delovni teden

Relacija: pogoste izdaje – stalna integracija?

1. Izbira **ene** uporabniške zgodbe za iteracij**e**
2. Identifikacija pod-zgodb (**inkrementi**)
3. Vsak inkrement gre skozi vse aktivnosti procesa (ena iteracij**a**)

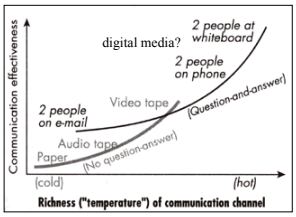
4. **Integracija** inkrementa
5. Ko vsi inkrementi v integriranem sistemu ustrezajo vsem testom sistema ⇒ majhna **izdaja**!

Načrtovanje izdaj



Osebna komunikacija > Dokumentacija

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



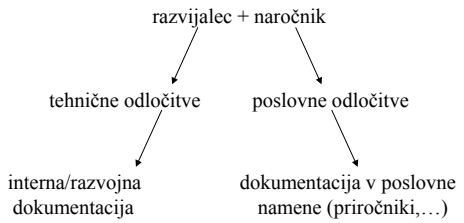
Delovno okolje

- Vsi v enem prostoru
- Že razporeditev v 2 nadstopji zmanjša komunikacijo
- Velik 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

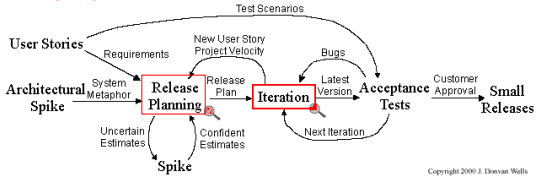


Odločitve o posamezni dokumentaciji so stvar prve ali druge skupine!

DODATEK #1

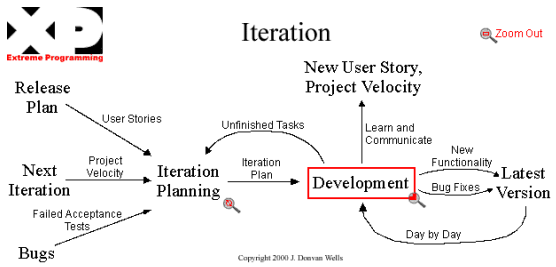
XP projekt

Extreme Programming Project



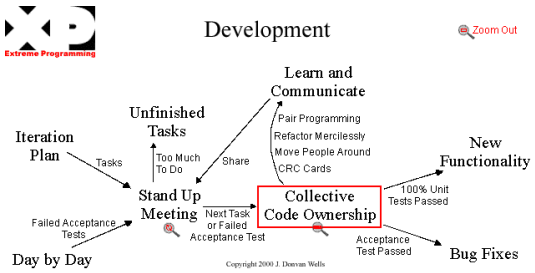
DODATEK #2

XP iteracija



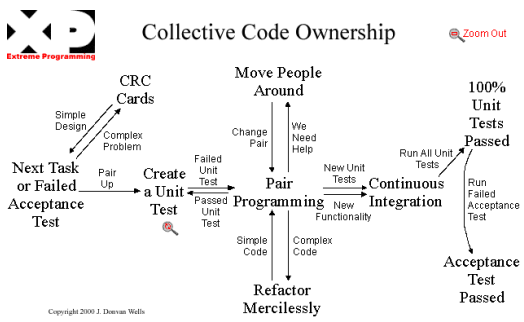
DODATEK #3

XP razvoj



DODATEK #4

XP skupno lastništvo kode



Unified **M**odeling **L**anguage

- **Modelirni** (**diagramski**) jezik, ki je neodvisen od razvojnega procesa (metode razvoja) in uporabljenih programskih jezikov v implementaciji!
- **5** pogledov na problem (projekt) da **9** diagramskih tehnik!

5 pogledov...

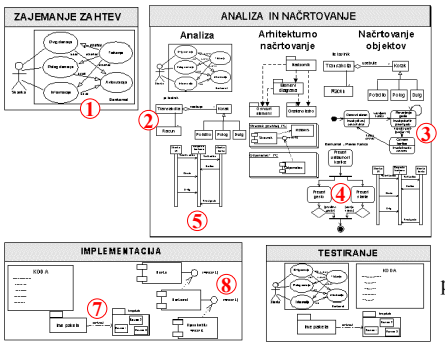
- Uporabniške zahteve (*diagram primerov uporabe*)
- Statična slika sistema (*razredni diagram*)
- Obnašanje programskega sistema (*diagram prehajanja stanj, diagram aktivnosti*)
- Interakcija objektov (*diagram zaporedja, diagram sodelovanja*)
- Implementacijski konstrukti (*komponentni diagram, paketni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

...in 9 tehnik

1. **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.
2. **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.
3. **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.
4. **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.
5. **diagram zaporedja** (sequence diagram);
   Prikazuje časovno (zaporedno) sodelovanje objekta v interakciji, ne prikazujejo pa asociacije med objekti.
6. **diagram sodelovanja** (interaction diagram);
   Prikazujejo obnašanje posameznega primera uporabe ter sodelovanje objektov v sklopu enega primera uporabe.
7. **paketni diagram** (package diagram);
   Predstavljajo skupine razredov,... in njihove odvisnosti (povezave) med seboj.
8. **komponentni diagram** (component diagram);
   Predstavljajo programske komponente in njihove odvisnosti (povezave) med seboj.
9. **diagram razvoja in dobave** (deployment diagram);
   Prikazuje fizične zveze (relacije) med programskimi in strojnimi komponentami sistema.

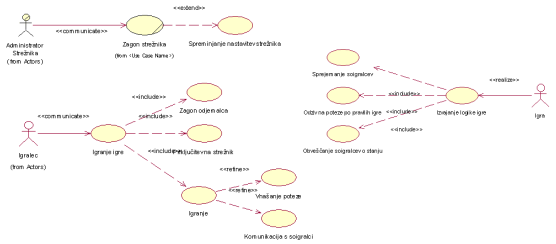
http://www.modelingstyle.info

Tipična uporaba posameznih diagramov

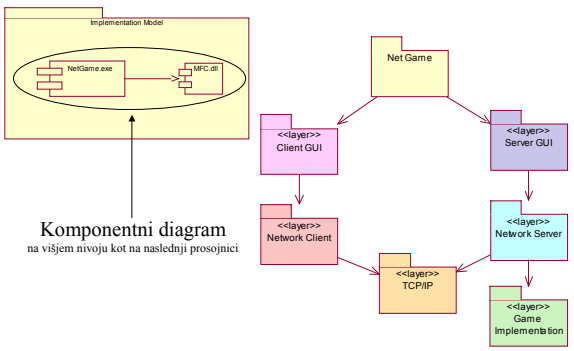


primeri posameznih diagramov ⇒

Diagram primerov uporabe

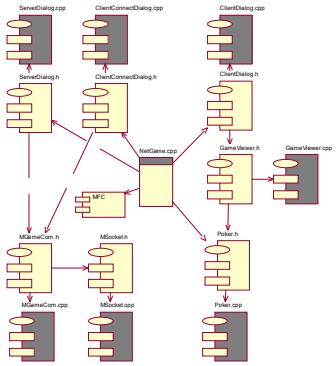


Paketni diagram



Komponentni diagram
na višjem nivoju kot na naslednji prosojnici

Komponentni diagram



Razredni diagram

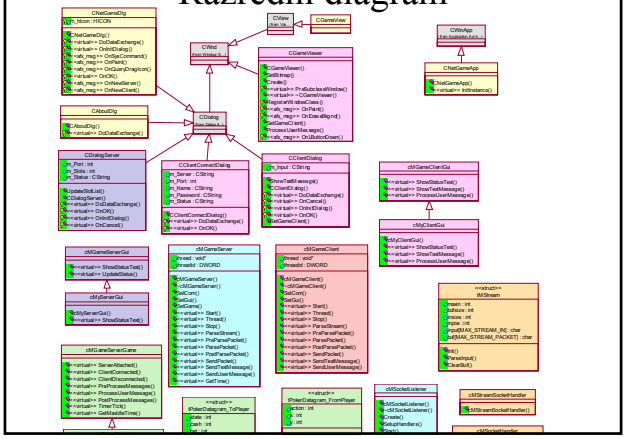


Diagram prehajanja stanj

Diagram zaporedja

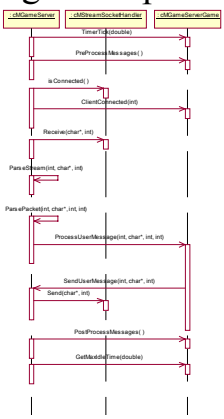


Diagram sodelovanja

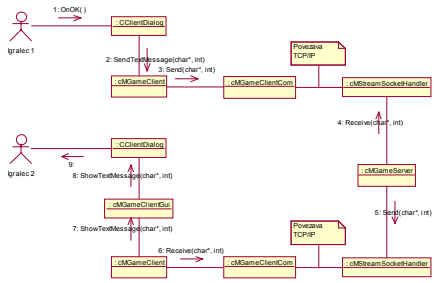


Diagram razvoja in dobave

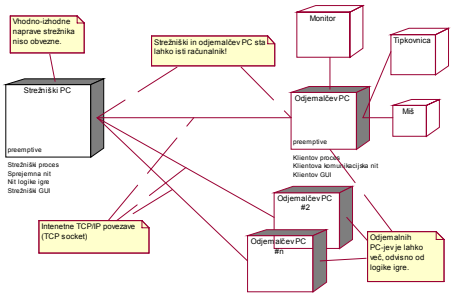
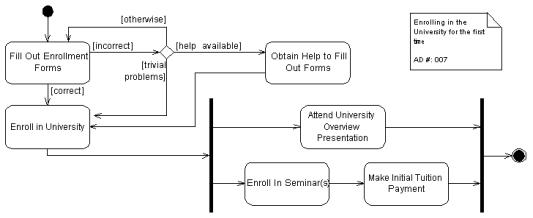


Diagram aktivnosti



Še pomnite: PowerBuilder/Designer?

(4GL RAD development tool / application modeling through: UML, Business Process Modeling techniques and traditional database modeling techniques)