UNIVERZITET DŽEMAL BIJEDIĆ U MOSTARU
FAKULTET INFORMACIJSKIH TEHNOLOGIJA MOSTAR

SEMINAR

Agilni *software development* *Toolbox* nove generacije developera

Student: *Ernad Husremović, DL 2792*
Mentor: *mr. Adil Joldić*

ver: 1.0.0

Mostar, decembar 2012.

SADRŽAJ

Abstract

Graphic user interface (GUI) je standardan interfejs modernih operativnih sistema. Donekle začuđujuće, nova - ”web generacija” developera intenzivno koristi Command-line interface (CLI) alate.
Ta skupina developera je na stanovištu da se u većini programerskih *Use-Case*-ova, maksimalna produktivnost najprije može postići uz pomoć ovih, po mnogo čemu ”old-fashion”[1] alata.
U ovom materijalu ćemo navesti ključne koncepte rješavanja problema programerske produktivnosti u predhodnih 20-tak godina računarstva.
Također, prikazaćemo glavne komponente programerskog seta ”novog” developera.
Pomenuti ”novi” developer je, gotovo u pravilu, ujedno i pobornih agilnih metoda razvoja software-a. Ta činjenica je od posebnog značaja u kontekstu šire teme koju autor izučava: *Agilni software development*.
Također, potražićemo uzroke takvom odabiru developera, polazeći od principa koje izučava *Human-computer interaction* (HCI).

Keywords: Programmer productivity, Agile software development, Regular expression, regex, regexp, Human-computer interaction, HCI, Command-line interface, CLI, GUI, vi editor, vim, vi macro, programmer's editor, code editor, CASE, IDE, REPL, javascript, java, ruby

[1] Većina ovih alata vuče korjenje iz prvih *Unix* sistema

1. Uvod

Human-computer interaction (HCI)[1] pročava zakonitosti interakcije čovjeka i računara. Koncept grafičkog interfejsa (GUI), miš (mouse), ekran osjetljiv na dodir (eng. touch screen) su dobro poznati rezultati izučavanja HCI-a.

1.1. Kategorizacija korisnika računara

Različite skupine korisnika imaju različite zahtjeve i potrebe u kontekstu HCI-a.
Tako malo dijete prepoznaje slike, ali ne razumije slova. Starijim i slabovidnim osobama potrebno je obezbjediti krupniji prikaz slova i znakova općenito. "Touch screen" je intuitivan ulazno-izlazni uređaj, čak i za dijete od tri-četiri godine. Tastatura to nije. Lahko je zaključiti da pripadnost određenoj *socijalnoj skupini* određuje moguće načine interakcije sa računarom.
Sljedeća bitna kategorizacija korisnika određena je poslovima koje korisnik obavlja na računaru.
U uputstvima za korištenje software-a se ovakva kategorizacija korisnika jednostavno naziva "ciljna grupa". Softverska rješenja su redovno složeni proizvodi, namjenjeni za *više ciljnih grupa* odjednom. Tako ćemo u uputstvima često naći sljedeće:

- pogodno za početnike ...
- Ova opcija je namjenjena isključivo naprednim korisnicima ...
- Pristup dozvoljen samo Administratoru sistema[2]

1.2. Produktivnost korisnika

Software sa stanovišta *ciljnog korisnika* treba biti *produktivan* alat. On treba pomoći da se određeni zadatak obavi brže i efikasnije nego li bi to bio slučaj bez njega.
Produktivnost pretpostavlja niz elemenata koje je potrebno zadovoljiti:

[1] Interakcija čovjek-računar
[2] Administrator sistema - stručna osoba zadužena za funkcionisanje IT sistema

- tačnost - *software* treba obezbjediti korsniku tačne rezultate
- fleksibilnost - software treba predvidjeti sve bitne varijante korištenja sa stanovišta korisnika
- robusnost - u slučaju problema i grešaka, gubici i zastoji korisnika trebaju biti minimalni

1.3. Potrebni period učenja software-a (eng. *Sofware Learning Curve*)

Poduktivnost "ne leži" samo u implementaciji softverskog rješenja. Produktivnost se postiže dobrim poznavanjem software-a od strane korisnika. Time dolazimo do još jedne bitne karakteristike software-a: Potrebni period učenja - *Learning curve*.[3]
Cilj je svakako da period učenja bude što manji. Ono što je veoma bitno, jeste postizanje napretka u početnom periodu učenja. Ukoliko korisnik ne stekne osjećaj postignuća u tom periodu, postoji velika mogućnost da će odustati od korištenja software-a (posebno ako su mu na raspolaganju alternativna rješenja).

1.4. Navike korisnika

Kada se korisnik osposobi za korištenje određenog softverskog rješenja, on prirodom stvari stiče navike. Prosječan korisnik mnoge stvari obavlja po *inerciji*. Čak i kada drugi korisnici postižu bolje rezultate, čak i kada su troškovi prelaska na druga rješenja minimalni, prosječan korisnik nije sklon promjenama software-a na koji je navikao. Stoga je, posebno u poslovnim primjenama, odabir odgovarajućeg rješenja veoma bitan korak.

1.5. Kategorizacije programera

Fokusirajmo se sada na programere kao posebnu ciljnu grupu. Sa stanovišta primarnih zaduženja, u domenu izrade poslovnih softverskih rješenja, adekvatna sljedeća podjela:

- aplikativni programer - zadužen za održavanje i realizaciju standarndih funkcija softverskog rješenja, usmjerenih ka krajnjem korsniku
- programer baznih funkcija (*"core libraries"*) - zadužen za niže nivoe arhitekture software-a

[3] Doslovni prevod na bio bi bosanski "krivulja učenja"

Tako će npr. aplikativni programer biti zadužen za izradu izvještaja ”Pregled dugovanja kupaca koji su prekoračili rok dospijeća obaveze”, dok će ”core” programer realizovati funkciju ”Razmjene podataka između prodavnica i centralne lokacije”.
Sa stanovišta vještina i iskustva, značajna je podjela:

- početnik - junior programer
- iskusni - senior programer

Softverska rješenja su složeni sistemi. Pojedine komponente traže dodatnu ekspertizu - specijalizaciju programera. To nas vodi ka sljedećoj kategorizaciji programera:

- database programer
- programer korisničkog interfejsa (UI)
- programer web rješenja
    - back-end
    - front-end

Ako posmatramo softverski projekat i njegov životni ciklus u cjelini, pored razvoja imamo i operacije podrške i instalacije[4]. Iz toga slijedi nova kategorizacija:

- poslovi podrške
- poslovi sistemske administracije
- poslovi razvoja i dizajna

Sve ove kategorizacije definišu definišu ciljne grupe *software*-a koje čini developerski *toolbox*.

1.6. Agilni software development

Agilni pristup se bazira na multi-funkcionalnim timovima. Članovi takvog tima su posljedično generalisti (Rasmusson, 2012, str. 19):

> *Kada formirate vaš tim, trebate generaliste, osobe koje nemaju problem raditi široki spektar stvari. Za programere, to znači naći osobe koje su sposobne raditi sa kompletnim tehnološkim stekom projekta (ne samo front-end ili back-end). Za testere i analiste, to znači osobe koje su spremne obaviti kako uobičajena testiranja, tako i detaljne analize korisničkih zahtjeva.*

[4]Često su u sve ove operacije programeri u većoj ili manjoj mjeri uključeni. Može se reći da je to čak pravilo ununtar manjih razvojnih timova

1.7. Programerska produktivnost

U predhodnih 20 godina, povećanje programerske produktivnosti se pokušalo postići na mnoge načine. Izdvojimo načine koji su posebno interesantni za ovo izlaganje:

- Izrada visoko sofisticiranih CASE[5] alata
- Integrirana razvojna okruženja (IDE)[6]

1.7.1. CASE alati

CASE alati su bili posebno popularan koncept kraja 80-ih i 90-ih godina (do ekspanzije interneta). CASE alati su pred sebe postavili iznimno ambiciozan cilj: napraviti okruženje koje će automatizirati kompletan životni ciklus razvoja software-a. Pri tome, zamišljeno je da se sam razvoj aplikacije realizira na visokom stepenu apstrakcije. Da bi se to postiglo, dizajneri CASE alata su trebali obezbjediti "kostur" aplikacije koji bi aplikativni programeri koristili kao polaznu tačku. Međutim, na kraju se pokazalo da tom konceptu nedostaje *fleksibilnost*. Koliko god bio dobro dizajniran, CASE alati nisu mogli predvidjeti sve moguće scenarije razvoja aplikacija. U konačnici, CASE alati su pružali visoki stepen produktivnosti u početku razvoja, ali se ta prednost ”topila” u kasnijim fazama razvoja aplikacija.
Drugi bitan faktor neuspjeha CASE koncepta je internet era, odnosno rapidan rast potrebe za www[7] rješenjima. CASE alati nisu mogli ponuditi kvalitetnu platformu za razvoj web rješenja[8].

1.7.2. IDE alati

IDE alati su, za razliku od CASE alata, standardni dio developerskog *toolbox*-a. Tu međutim treba izvodijit jednu opciju koja je u početku smatrana ključnom, da bi na kraju postala marginalizirana. To je funkcija Vizuelnog dizajnera formi (eng. Visual Forms Designer). Prve generacije IDE alata su ovu funkciju stavljale na vrh svojih *Feature* listi. Slično CASE alatimima, vizuelnog programiranje formi se pokazalo neadekvatnim kada aplikacija dosegne određeni nivo kompleksnosti. Kada se to desi, vizuelni mod razvoja se napušta i programer programira korisnički interfejs direktno u kôdu. Također, ”Visual Forms” alati do sada nisu postigli veliku primjenu na području razvoja web aplikacija, što je negativno išlo u prilog konceptu ”vizuelnog programiranja”.

[5] `http://en.wikipedia.org/wiki/Computer-aided_software_engineering#Tools`
[6] `http://en.wikipedia.org/wiki/Integrated_development_environment`
[7] `http://hr.wikipedia.org/wiki/World_Wide_Web`
[8] Web standardi su bili u povoju. Arhitektura web rješenja je u početku bila nejasna, fagmentacija web browsera velika

IDE okruženja su zato fokus razvoja usmjerili na produktivnost programera unutar *source code* editora. Ključne opcije postaju funkcije koje asistiraju programeru:

- automatsko kompletiranje imena funkcija, varijabli, klasa[9]
- sugestije i pomoć programeru prilikom pisanja izvornog koda
  - pomoć pri formatiranju kôda - autoindent
  - provjera sintakse - *syntax checking*
- integrirano debugiranje

1.8. Programski jezici

1.8.1. Java kao univerzalni programski jezik

"Java"[10] programski jezik su početkom 90-tih mnogi smatrali univerzalnim programskim jezikom. Na kraju, ta očekivanja se nisu ispunila. Posebno su se pokazali nedostatnim sljedeće osobine "java"-e[11]:

- java na strani web klijenta (java web plugin sistem)
- kompleksnost java serverskog steka J2EE[13]

U posljednjim implementacijama J2EE[14] su napravljeni značajni pomaci. Međutim, veliki broj developera je u međuvremenu prihvatio alternativne tehnologije i programske jezike.

1.8.2. Dinamički programski jezici

Ekspanzija zahtjeva za web i mobilnim rješenjima je rezultiralo pravom "invazijom" programskih jezika.
Nova generacija developera, posebno u domenu razvoja web aplikacija, usvaja dinamičke programske jezike[15].
Jedan od ključnih faktora tog trenda je značajno skraćen *Learning curve* u odnosu na konvencionalne platforme (J2EE, .NET).

---

[9] Microsoftova implementacija `http://en.wikipedia.org/wiki/IntelliSense`

[10] `http://en.wikipedia.org/wiki/Java_(programming_language)`

[11] Preciznije je reći: "Osobine JVM platforme" s obzirom da je "java" samo jedan od jezika koji se izvršavaju na JVM[12] platformi

[13] `http://radio-weblogs.com/0135826/2004/03/17.html`

[14] `http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition`

[15] `http://en.wikipedia.org/wiki/Dynamic_programming_language`

1.8.3. Javascript, *"Lingua franca"* web-a

Ekspanzija web-a je dovela do neslućene ekspanzije "Javascript" programskog jezika[16]. Primarno zamišljen kao programski jezik *web browsera*, javascript se počinje primjenjivati i van browsera:

- mobilni klijent - appcelerator titanium[17]
- *server-side* - node.js[18]
- *database stored procedure* - plv8[19]

1.9. Efekat masovnog usvajanja programske platforme

Tezu o "java"-i kao univerzalnom programskom jeziku, zamijenila je teza o "javascript"-u kao de-facto univerzalnom jeziku web-a[20]. Cilj ovog materijala nije dokazati ili osporiti ovu tezu.

Međutim, svakako je bitno uočiti značaj usvajanja određene platforme, odnosno programskog jezika od strane velike grupe korisnika.

Web browser i *Internet* općenito su postali univerzalna računarska platforma.

To je razlog što je nova generacija developera prigrlila javascript/HTML/CSS. To je razlog što nova generacija programerskih alata želi obezbjediti visok stepen integracije i/ili korištenje ovih tehnologija[21].

Ovaj developerski trend je doslovno natjerao najveće IT kompanije na strateške promjene u posljednje dvije godine[22]:

- "Microsoft" stavlja fokus na HTML5 pored sopstvene RIA[23] platforme "Silverlight"[24]
- "Adobe" se također usmjerava na HTML5, pored sopstvene Flash platforme[25]

---

[16] http://en.wikipedia.org/wiki/JavaScript

[17] http://www.appcelerator.com/platform/titanium-sdk

[18] http://nodejs.org

[19] http://code.google.com/p/plv8js/wiki/PLV8

[20] http://www.codinghorror.com/blog/2007/05/javascript-the-lingua-franca-of-the-web.html

[21] http://wiki.developerforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

[22] Ako su se ove kompanije odlučile "odreći" platformi koje su im do sada obezbjeđivale dominaciju na tržištu, onda svaki drugi učesnik u IT industriji treba, ako ništa drugo, dobro razmisliti o svojoj budućoj strategiji razvoja

[23] http://en.wikipedia.org/wiki/Rich_Internet_application

[24] Trijumf HTML5 nad zatvorenim vendorskim tehnologijama

[25] http://www.adobe.com/products/flash/flash-to-html5.html

2. *Toolbox* nove generacije developera

Dinamički programski jezici (php, python, ruby, javascript) su generirali potrebe za razvojem nove generacije programerskih alata.
Tu se pojavio značajan problem. Niz funkcija koje je relativno jednostavno implementirati kod *strong-type*[1] programskih jezika[2] je teško realizovati kod dinamičkih jezika.

2.1. Povratak u prošlost - komandna linija i programerski editor

Nova generacija developera se ne može osloniti na postojeće alate i prakse. Prisiljeni su tragati za alternativnim rješenjima. Njihov odabir je krajnje interesantan. Oni u svakodnevnom radu intenzivno koriste ”starinske” alate:

- Umjesto IDE okruženja, koriste programerske editore[3] vi, emacs[4]
- intenizvno koriste terminal i komandnu liniju[5]

2.2. TDD umjesto *debuggera*

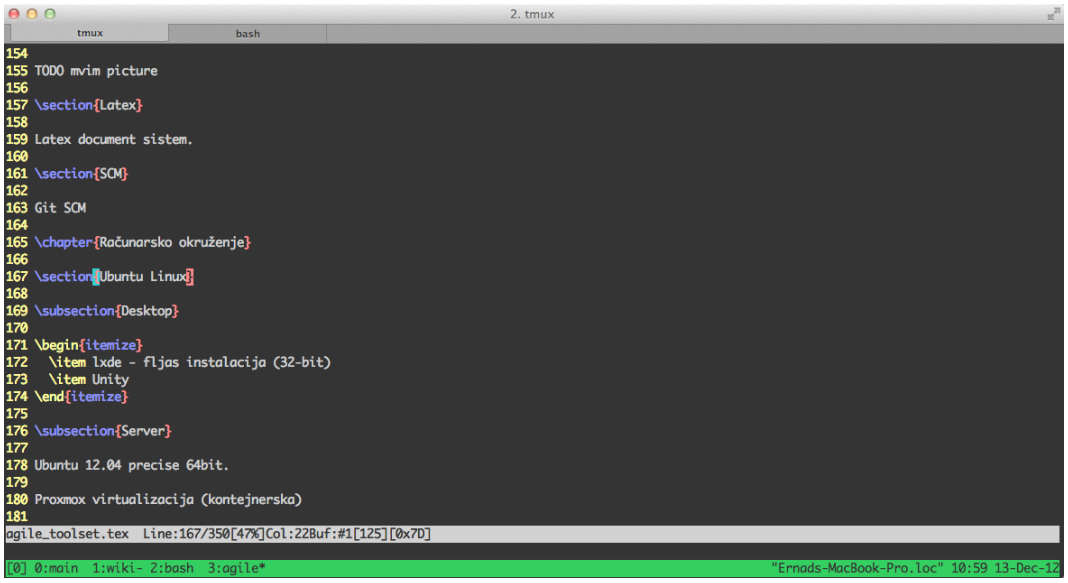
Podrške interaktivnom debugiranju dinamičkih jezika je, posebno u početku, bila na niskom nivou. Taj nedostatak je značajno uticao na pojavu nove paradigme programiranja: *Test driven development* - (TDD)[6].
TDD se brzo pokazao kao pozitivna programerska praksa koja utiče na povećanje kvaliteta izvornog kôda neovisno od nivoa podrške debugiranju.

[1] `http://en.wikipedia.org/wiki/Strong_typing`
[2] Dobar je primjer *autocomplete* funkcija IDE-a
[3] `http://net.tutsplus.com/tag/code-editors/`
[4] Oni konvencinalniji Notepad++, TextMate, jEdit
[5] `http://en.wikipedia.org/wiki/Strong_typing`
[6] `http://en.wikipedia.org/wiki/Test-driven_development`

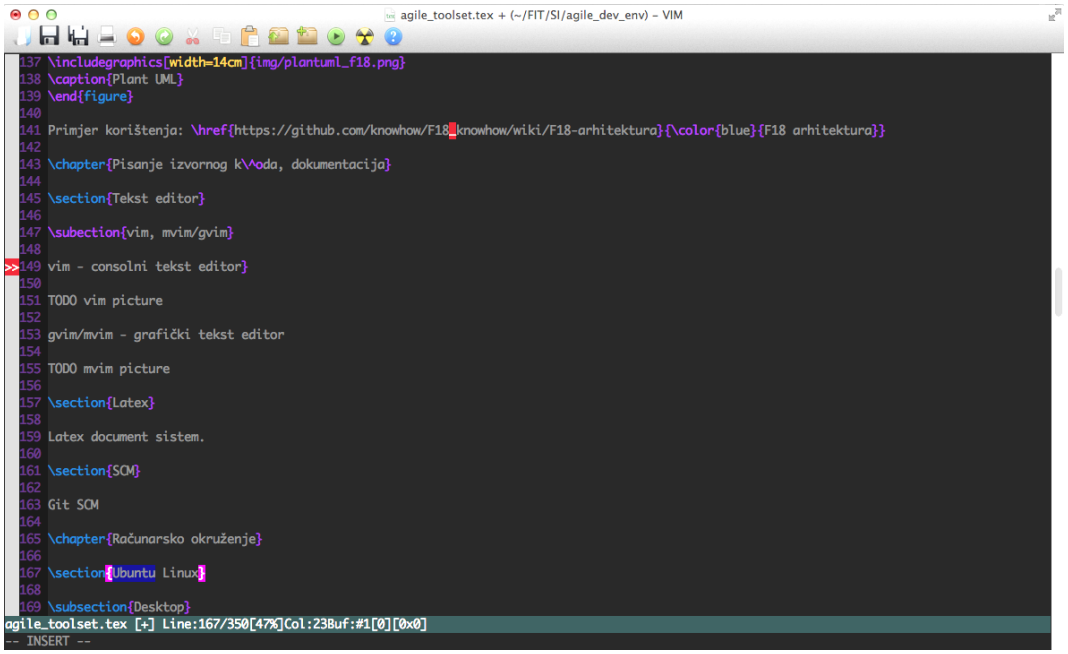
2.3. Programerski editor umjesto IDE-a

2.3.1. *Vi* editor

Vi editor[7] je svojom popularnošću kod nove generacije developera[8] po mnogo čemu fenomen savremene istorije razvoja software-a.



Slika 2.1: vim, terminal verzija



Slika 2.2: mvim, Mac OS X grafička verzija vim editora

[7] http://en.wikipedia.org/wiki/Vim_(text_editor)
[8] https://github.com/tpope, http://blog.sanctum.geek.nz/vim-koans/

Vi editor je neintuitivan za početnika. *"Learning curve"* ovog editora je iznimno nepovoljan. *Vi* ima sasvim drugačiji koncept korištenja u odnosu na konvencionalne editora. Sa druge strane, *vi* se uvijek nalazi na listi najpopularnijih programerskih editora[9].

2.3.2. *Vi* - editor koji "misli" kao programer

Autor ovog teksta je često i sam isfrustriran *vi*-om. Moji razlozi za usvajanje *Vi* su bili sljedeći:

- podrška linux serverima bez GUI interfejs-a
- Linux, Mac, Windows desktop - dostupnost GUI (gvim, mvim) ili terminal varijante, po izboru
- najbolja podrška za *harbour*[10]
- brz rad i na računarima sa minimum RAM-a[12]

Ukratko, *vi* je obavljao ono što trebam. Međutim, nikada nisam uspio do kraja usvojiti "vi-način" korištenja. Pripremajući se za ovaj rad nabavio sam knjigu "Practical Vim, Edit Text at the Speed of Thought"(Neil, 2012).
Čitajući knjigu brzo sam došao do sljedećeg zaključka:

- Nakon 10-tak godina rada sa *vi*-om, ja sam i dalje *vi beginner*
- Po svoj prilici, nikada neću ni postati *vi master*[13]

Da bi se postalo *vi masterom*, treba puno vježbe. Također, treba imati bistar um, um otvoren za usvajanje novih vještina.
Učenje i istraživanje *vi*-a će sigurno pomoći (posebno mlađem) programeru da podigne svoje vještine na viši nivo. To učenje će biti od koristi bez obzira da li će *vi* biti primarni editor ili ne. *Vi* "tjera" korisnika na *rethink* svakodnevnih operacija pisanja programskog kôda. On pokazuje da uvijek postoji drugačiji i efikasniji način da se one obave. Mišljenja sam da je usvajanje tog *rethink* pristupa ključni preduslov na putu ka **izvrsnosti** u programerskom poslu.
Bez obzira na ranije zaključke, *vi* ostaje moj izbor programerskog editora. Koliko god često se osjećao "glupim" pred ekranom *vi*-a, i dalje ostaje zabavno otkrivati njegove mogućnosti i nove načine korištenja. Dodatno, pitanje developerskog *toolset*-a nije samo lično pitanje programera, nego pitanje od interesa za čitav razvojni tim. Ako ja i ne mogu postati *vi*

[9] Pregled programerskih editora
[10] [11]
[12] Ovaj argument postaje sve manje bitan, ali i dalje postoje postojati uređaji koji imaju ograničene resurse - npr. `https://openwrt.org`
[13] `http://blog.sanctum.geek.nz/vim-koans`

master, drugi članovi mogu. Novim kolegama ipak mogu pomoći, posebno na početku, da bolje razumiju osnovne koncepte *vi*, da počnu razmišljati na "vi-način". Zato unutar razvojnog tima u kome djelujem svoje kolege potičem na korištenje *vi*-a. To i jeste jedan od glavnih motiva za pisanje ovog rada.

2.3.3. *Vi* - primjer korištenja

Dugo sam razmišljao o tome šta bi to moglo demonstrirati "vi-način" pisanja programskog kôda, a da pri tome bude razumljivo čitaocu koji se po prvi put susreće sa njim.
Na kraju sam se sjetio jednog primjera od prije par mjeseci. Zadatak je napraviti unos HTML "select"-a elementa radi unosa godine rođenja[14].
Demonstriraćemo dva načina rješavanja ovog zadatka. Krenismo sa prvom varijantom.
Pokrenite *vi* i unesite sljedeći tekst[15]

```
1 <html>
2 <body>
3 Godina rodjenja:
4 <select name="datum_rodjenja">
5   <option value="">Odaberi godinu</option>
6   <option value="1910">1910</option>
7 </select>
8 </body>
9 </html>
```

Onda ukucajte sljedeće:

Kucati:	Opis
`<ESC>`	pređimo u "Normal mode" za svaki slučaj
`:set number<ENTER>`	prikažimo redne brojeve radi preglednosti
`6gg`	skočimo na liniju 6

Nakon ovoga smo na poziciji drugog "option" elementa. Zastanimo malo. Razmislimo. Šta trebamo uraditi da dođemo do cilja ?

- Trebamo iskopirati tekuću "option" liniju
- U toj novoj liniji trebamo povećati godinu na oba mjesta
- onda to trebamo ponoviti cca 100 puta

[14] Striktno gledajući, jedino dobro rješenje je napraviti dinammičko kreiranje takve kontrole kao što je to realizovano ovdje. S obzirom da tražimo brzo rješenje, tu opciju isključujemo

[15] Ili kopirajte ovaj gist

2.3.4. *Don't Repeat Yourself* (DRY) princip i *vi*

Vi "macro" sistem nam omogućava da gornje operacije realizujemo poštujući DRY princip. Napravićemo našu prvu *vi* makro komandu:

Kucati:	Opis
`qa`	započni snimanje *vi* makroa u "a" registar
`0`	idi na početak linije
`yy`	snimi ("yank") tekuću liniju u registar
`p`	kreiraj ("paste") novu liniju tako što ćeš je uzeti iz tekućeg registra (registar napunjenom predhodnom "yank" operacijom)
`<C-a>`	<Ctrl> + 'a' - na tekućoj liniji pronađi broj i uvećaj ga za 1 (add operator, <C-x> je minus operator)
`w`	otiđi na sljedeću riječ (pomjeri sa pozicije prvog broja)
`<C-a>`	povećaj i drugi broj
`yy`	ponovo snimi tekuću liniju u registar (da bi naredni paste uze posljednje uvećane brojeve)
`q`	završi snimanje makroa.

Stavimo sve iz gornje tabele u jednu sekvencu komandi: `qa0yyp<C-a>w<c-A>yyq`.
Testirajmo naš makro sa `@a` (*replay* makro "a").
Trebamo dobiti novu liniju sljedećeg sadržaja: `<option value="1911">1911</option>`.
Na kraju, zadajmo *vi*-u da istu operaciju ponovi još 101 put. Ukucajmo: `101@a`.
Na kraju dobijamo konačni rezultat:

```
<html>
<body>
Godina rodjenja:
<select name="datum_rodjenja">
  <option value="">Odaberi godinu</option>
  <option value="1910">1910</option>
  <option value="1911">1911</option>
    ....

  <option value="2012">2012</option>
</select>
</body>
</html>
```

Zaključujemo: *vi master* (vidi 2.3.2) ovakav zadatak doista može realizovati "brzinom misli".

2.3.5. Realizacija zadatka na drugi način: *vi* + dinamički jezik

Uzmimo dinamički jezik koji poznajemo. U njemu ćemo napraviti program koji će izgenerisati željeni tekst. Ruby program izgledao bi ovako:

```
#!/usr/bin/env ruby
puts "<select␣name=\"datum_rodjenja\">"

(1910..2012).each do |i|
     puts "␣␣␣<option␣value=\"#{i}\">#{i}</option>"
end

puts "</select>"
```

Slika 2.3: ruby generator teksta (*/tmp/gen_options.rb*)

Sa `r` komandom učitavamo izlaz iz programa (ono što program pošalje na `stdout` uređaj) u tekući dokument[16]:

```
:r !ruby /tmp/gen_option.rb
```

Uočavamo da je prva varijanta efikasnija kako sa stanovišta utroška vremena, tako i potrebne energije programera da dođe do cilja[17].

2.3.6. IDE na *Command-line* interfejsu

U svjetlu ove teme interesantno je proučiti sljedeće pisane i video materijale:

- Unix kao IDE
- *Remote* "Pair programming" sa "tmux"-om[18]

"Pair programming"[19] je inače jedna od ključnih developerskih praksi po XP metodologiji(J.Shore i S.Warden, 2008).

2.4. Dolazeća generacija programerskih alata

"Javascript everywhere" princip širi se i na područje programerskih alata:

[16]":" ukazuje na unos u komandom modu *vi*-a. Naravno, da bi ovo funkcionisalo *ruby* interpreter postojati na sistemu

[17]Pod uslovom da je programer dobro uvježban i sa *vi*-om i sa *ruby*-jem (odnosno dinamičkim jezikom po izboru)

[18]`http://tmux.sourceforge.net`, te knjiga "tmux: Productive Mouse-Free Development"(Hogan, 2012)

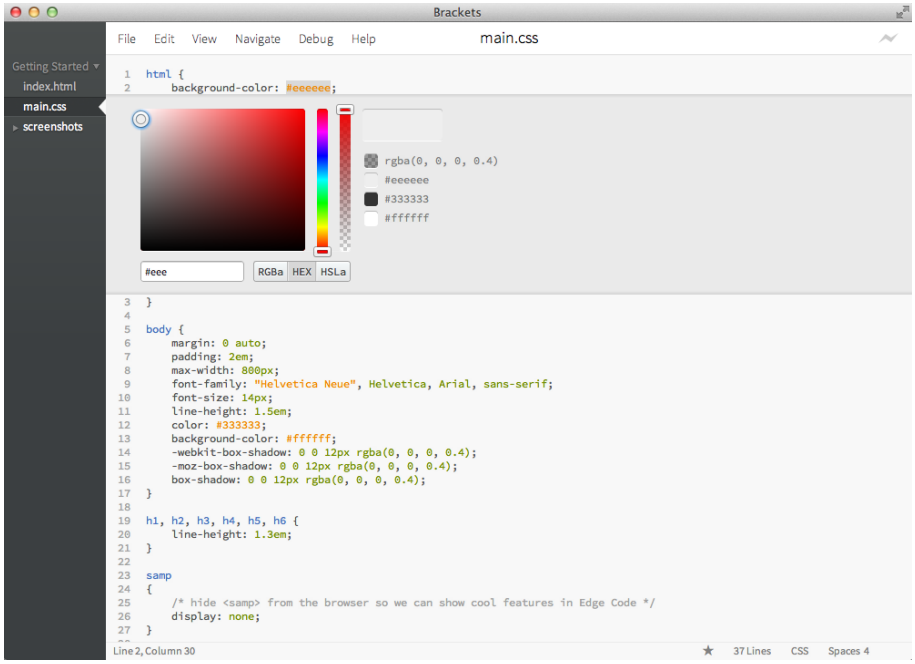
[19]`http://en.wikipedia.org/wiki/Pair_programming`

– *Adobe Brackets code editor* - `http://brackets.io`

– *online* - alati u *cloud*-u:

- *CloudIDE* - `https://c9.io`
- *Github code snippets* - `https://gist.github.com`

Video nettuts+ ”A Peek At Brackets”[20] ukazuje na inovativni pristup produktivnom pisanju Javascript/HTML/CSS kôda.



Slika 2.4: Brackets code editor - *quick edit* ”color” elementa

Programerski alati u *cloud*-u je drugi značajan trend u ovoj oblasti. Socijalni aspekt *online* alata - pojednostavljena komunikacija između developera su bitan katalizator ovih promjena.

[20] `http://www.youtube.com/watch?v=HZkrlX7jJcg`

3. Komandna linija

"Novi developer" svakodnevno koristi terminal[1] i komandnu liniju. U ovom poglavlju ćemo prikazati najbitnije načine korištenja *command-line* alata.

3.1. *Command-line* interfejs (CLI) sa stanovišta HCI-a

Prednosti *Command-line* interfejsa[2]:

- brz i moćan za iskusne korisnike
- minimalna količina tipkanja (miš se ne koristi)
- može se koristiti u kombinaciji sa drugim korisničkim interfejsima

Nedostaci:

- izvršenje komandi sa malo ili bez pitanja korisnika
- traži dobro poznavanje sistema i programa
- bazira se na sjećanju - memorisanju komandi i sintakse
- težak za učenje
- sklon greškama

3.2. *Unix Pipe*

Unix Pipeline[3] je jednostavan, ali nadasve moćan koncept kreiranja složenih komandi preusmjeravanjem izlaza predhodne u ulaz naredne komande.

[1] `http://en.wikipedia.org/wiki/Terminal_emulator`
[2] `http://www.cs.man.ac.uk/~seanb/teaching/COMP10092/COMP10092-HCI.pdf`
[3] `http://en.wikipedia.org/wiki/Pipeline_(Unix)`

```
$ ps ax | grep MacVim | grep -v grep
```

```
 585 ?? S 1:13.89 /Applications/MacVim.app/Contents/MacOS/MacVim -↩
    psn_0_249917
86785 ?? Ss 0:00.23 /Applications/MacVim.app/Contents/MacOS/Vim -g -f ↩
   --mmwaitforack
```

Slika 3.1: Pregled svih "MacVim" trenutno aktivnih procesa

3.3. *Read-eval-print-loop* (REPL) konzola

Dinamički jezici redovno posjeduju odgovarajuću REPL[4] konzolu. REPL konzola omogućava interaktivni unos komandi dinamičkog programskog jezika.
Mogućnosti primjene su raznovrsne - od testiranja do ad-hoc primjena kao što je to demonstrirano u sljedećem primjeru.

3.3.1. Programerski kalkulator bez limita - *irb*

Ruby[5] REPL konzola naziva se `irb`.

```
$ irb
```

```
$ irb
irb> 228 % 15 # cjelobrojni ostatak
 => 3
irb> 2**3 # stepenovanje
 => 8
irb> a = 2**3 + 528/5.2
 => 109.53846153846153
irb> b = Math.sin(0.2) + 2 * Math.cos(0.8)
 => 1.5920827494893923
irb> a + 2.2 / b
 => 110.92029926059348
irb> # http://stackoverflow.com/questions/84421/converting-an-integer-↩
   to-a-hexadecimal-string-in-ruby
irb> "%x" % (0x0FF + 0x0EE)
=> "1ed"
```

Slika 3.2: ruby "irb" - kalkulator bez limita

[4] http://en.wikipedia.org/wiki/Read\T1\textendasheval\T1\textendashprint_loop

[5] http://www.ruby-lang.org

3.4. *Regular expressions*

Regular expressions[6] (regexp) se koriste kod mnogih CLI alata. Takođe, moderni programski jezici imaju ugrađenu podršku za *regex*-e:

- ruby podrška[7]
- grep *text search* alat[8]
- sed *stream editor*[9]
- awk[10]
- git grep (vidi 3.5.1)
- vim plugin "CtrlP" za brzu pretragu fajlova po imenu[11]

Poznavanje i korištenje *regex*-a je bitan faktor produktivnosti programera.

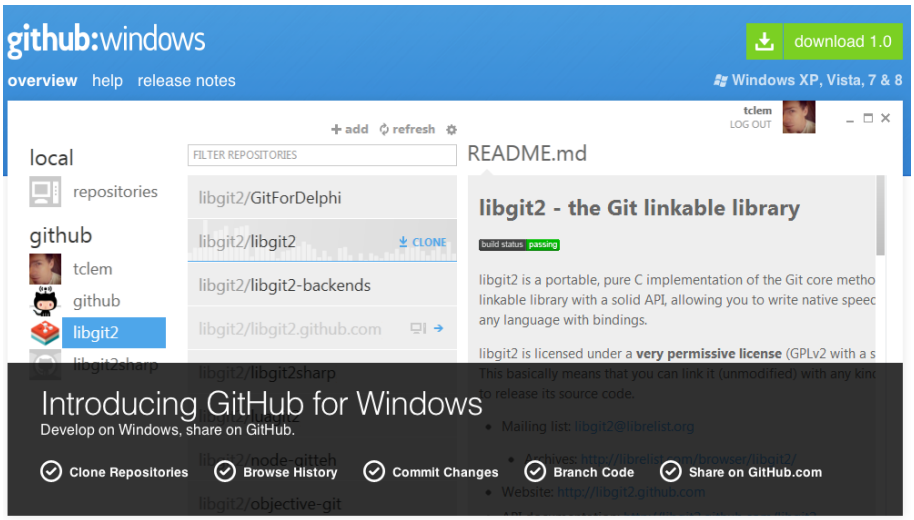
3.5. *Command-line interface* (CLI) aplikacije

CLI aplikacije se navode na komandnoj liniji - *(*shell)-u operativnog sistema[12].

3.5.1. Primjer: "git" klijent

Git je popularni SCM alat[13]
Postoji niz git klijenata sa GUI interfejsom[14].

[6] `http://en.wikipedia.org/wiki/Regular_expression`
[7] `http://www.ruby-doc.org/core-1.9.3/Regexp.html`
[8] `http://en.wikipedia.org/wiki/Grep`
[9] `http://en.wikipedia.org/wiki/Sed`
[10] `http://en.wikipedia.org/wiki/AWK`
[11] `https://github.com/kien/ctrlp.vim`
[12] Windows: cmd, PowerShell, Unix: sh, bash, zsh
[13] Detaljnije u Git SCM (Husremović, 2012)
[14] http://git-scm.com/downloads/guis



Slika 3.3: Naslovna stranica github GUI klijenta za Windows OS

Ipak, većina git korisnika koristi osnovnu, CLI verziju:

```
~$ git --help

=>

usage: git [--version] [--exec-path[=<path>]] [--html-path] [--man-path↩
   ] [--info-path]
        [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        [-c name=value] [--help]
        <command> [<args>]

The most commonly used git commands are:
  add Add file contents to the index
  bisect Find by binary search the change that introduced a bug
  branch List, create, or delete branches
  checkout Checkout a branch or paths to the working tree
  clone Clone a repository into a new directory
  commit Record changes to the repository
  diff Show changes between commits, commit and working tree, etc
  fetch Download objects and refs from another repository
  grep Print lines matching a pattern
  init Create an empty git repository or reinitialize an existing one
  log Show commit logs
  merge Join two or more development histories together
  mv Move or rename a file, a directory, or a symlink
  pull Fetch from and merge with another repository or a local branch
  push Update remote refs along with associated objects
  rebase Forward-port local commits to the updated upstream head
  reset Reset current HEAD to the specified state
  rm Remove files from the working tree and from the index
  show Show various types of objects
  status Show the working tree status
  tag Create, list, delete or verify a tag object signed with GPG

See 'git help <command>' for more information on a specific command.
```

Slika 3.4: git CLI klijent

Zašto developeri napuštaju "komfor" GUI okruženja ? Zato što je CLI, kada se njime ovlada, efikasniji interfejs (vidi: 3.1).
To je razlog zbog koga su se GUI alati uglavnom orjentisali na operacije najčešće korištene operacije, te operacije kod kojih je grafički prikaz evidentna prednost. U konkretnom slučaju,

GUI klijenti su pogodniji za prikaz istorije promjena ili razlika između pojedinih verzija[15] CLI klijent omogućava da se određene operacije obave na drugačiji način. Ako želimo pretražiti istoriju promjena, iskusni korisnik će prije posegnuti za `git grep` komandom, umjesto GUI prikaza istorije promjena:

```
NAME
     git-grep - Print lines matching a pattern

SYNOPSIS
     git grep [-a | --text] [-I] [-i | --ignore-case] [-w | --word-↩
        regexp]
             [-v | --invert-match] [-h|-H] [--full-name]
             [-E | --extended-regexp] [-G | --basic-regexp]
             [-P | --perl-regexp]
             [-F | --fixed-strings] [-n | --line-number]
             [-l | --files-with-matches] [-L | --files-without-match]
             [(-O | --open-files-in-pager) [<pager>]]
             [-z | --null]
             [-c | --count] [--all-match] [-q | --quiet]
             [--max-depth <depth>]
             [--color[=<when>] | --no-color]
             [-A <post-context>] [-B <pre-context>] [-C <context>]
             [-f <file>] [-e] <pattern>
             [--and|--or|--not|(|)|-e <pattern>...]
             [ [--exclude-standard] [--cached | --no-index | --↩
                untracked] | <tree>...]
             [--] [<pathspec>...]
```

[15] Napomenimo da iskusni korisnici, ne preferiraju *switch*-anje između GUI i CLI klijenta. Operacija *switch*-anja sama po sebi narušava produktivnost.

4. Zaključak

Programerska produktivnost je vječita tema IT-a.
Programer, IT profesionalac općenito, je po mnogim aspektima posebna kategorija korisnika računarskog sistema.
Visok nivo apstrakcije (GUI) u korištenju računarskog sistema pomaže korisniku da lakše rukuje računarskim sistemom. Visok nivo apstrakcije međutim nužno smanjuje fleksibilnost u radu korisnika i moguće načine korištenja. U mnogo slučajeva ta fleksibilnost i nije potrebna.
Za razliku od "običnog" korisnika, kod programera je potreba za fleksibilnošću puno izraženija. Svakodnevne operacije programera su pune repetitivnih operacija. **Automatizacija** takvih operacija nerijetko biva značajan diferencijator između uspješnog i neuspješnog razvojnog tima.
Programer je prirodom svog sposla sposobniji komunicirati sa računarskim sistemom na nižem nivou apstrakcije (CLI interfejs). Ukratko, programer može maksimalno iskoristiti prednosti CLI-a (vidi 3.1), a da mu pri tome evidenta ograničenja CLI-a ne budu nepremostiv problem (nepovoljan "Learning Curve").
Niz je primjera u kojima "novi web developer" uvodi značajne inovacije na planu programerske produktivnosti i koncepta softverskih rješenja općenito. Novi develoeper je svjestan da do tih rješenja nije moguće doći bez promjena paradigme u pristupu razvoju software-u.[1]
Internet tehnologije i otvoreni standardi su temelj tih inovacija.

[1]Nakon tri gotine studija na FIT-u, nisam uočio da se ovim temama pridaje potrebna pažnja. To me je dodatno motiviralno na izradu ovog rada (primarni motiv - vidi 2.3.2).

5. Literatura

Brian P. Hogan. *tmux: Productive Mouse-Free Development*. The Pragmatic Programmers, 2012.

Ernad Husremović. *Agilni software development, Git SCM*, 2012. URL `https://github.com/hernad/agile_dev_env/raw/master/agile_git.pdf`.

J.Shore i S.Warden. *The Art of Agile Development*. O'Reilly, 2008.

Drew Neil. *Practical Vim, Edit Text at the Speed of Thought*. The Pragmatic Programmers, 2012.

Jonanthan Rasmusson. *The Agile Samurai (P4.0)*. The Pragmatic Programmers, 2012.