

UNIVERZITET DŽEMAL BIJEDIĆ U MOSTARU
FAKULTET INFORMACIJSKIH TEHNOLOGIJA

Završni rad nakon I ciklusa

Agilni razvoj softvera

Student: Ernad Husremović, DL 2792

Mentor: prof. dr. Emina Junuz

Mostar, septembar 2018.

(posveta todo)

Sadržaj

Sažetak.....	6
1 Uvod.....	7
1.1 Manifest agilnog razvoja softvera.....	7
1.2 Osnovni principi u agilnom razvoju softvera.....	7
1.3 Usporedba tradicionalnih i agilnih metoda razvoja.....	8
1.3.1 Tradicionalne metode.....	8
1.3.2 Agilne metode.....	9
2 Ekstremno programiranje (XP).....	10
2.1 Uvod.....	10
2.2 Organizacija XP tima.....	10
2.3 Restruktuiranje - refactoring programskog kôda.....	11
2.4 Programiranje u paru.....	11
2.5 Testiranje na prvom mjestu (TDD).....	11
2.5.1 Rust TDD i refactoring primjer.....	11
3 “Scrum”.....	16
3.1 Uvod.....	16
3.2 Dnevni “Stand-up” sastanci (eng. stand-up meetings).....	16
3.3 “Sprintovi”.....	17
3.3.1 Centralna lista korisničkih priča (eng. Product backlog).....	17
3.3.2 Lista priča novog sprinta (eng. Spring backlog).....	17
4 “Lean software development”.....	18
4.1 Uvod.....	18
4.2 Eliminacija otpada.....	18
4.3 Kvalitet izrade.....	18
4.3.1 Kreiranje znanja.....	19
4.3.2 Donošenje odluka u najboljem trenutku.....	19
4.3.3 Brze isporuke.....	19
4.3.4 Uvažavanje ljudi.....	19
4.3.5 Optimiziranje cjeline.....	19
5 “Kanban”.....	20

5.1 Uvod.....	20
5.2 “Kanban” ploča.....	20
6 Zajedničke prakse i koncepti.....	23
6.1 Upravljanje revizijama izvornog kôda (VCS).....	23
6.1.1 Github repozitorij F18 projekta.....	25
6.2 Multi-funkcionalni tim.....	26
6.3 Agilni trener.....	27
7 F18 - “knjigovodstvo za Bosance”.....	28
7.1 Uvod.....	28
7.1.1 Cilj projekta.....	28
7.1.2 “Ad-hoc” rješenja, dugoročne glavobolje.....	29
7.2 Preduzeće “bring.out”.....	29
7.2.1 Razvojni period, developeri.....	29
7.2.2 Kriza projekta.....	30
7.3 Korištene tehnologije.....	30
7.3.1 Harbour programski jezik.....	30
7.3.2 Pristup PostgreSQL bazi iz “harbour”-a.....	31
7.3.3 Više-platformski pristup.....	34
7.4 Upravljanje projektom.....	34
7.4.1 Poslovni kontekst projekta.....	34
7.4.2 Upotrebljena vrijednost.....	34
7.5 Git istorija projekta 2014-2018.....	35
7.6 Glavni ciljevi razvoja “F18” u periodu 2016-2018.....	36
7.6.1 Čitljivost izvornog kôda (eng. source code readability).....	36
7.6.2 Promjene na sloju pristupa podataka (eng. data layer).....	36
7.6.3 Realizacija programskih zahtjeva korisnika.....	36
7.7 Pregled F18, korisnički nivo.....	37
7.8 Postavke - parametri F18.....	37
7.9 FIN - Finansijsko poslovanje.....	38
7.10 FAKT - modul za fakturisanje.....	40
7.11 Ostali programski moduli.....	42
8 Razvoj projekta “F18” u periodu 2014-2018.....	43
8.1 Uvod.....	43

8.2 “Atom” editor i podrška za “harbour”.....	43
8.2.1 Atom “language-harbour” ekstenzija.....	44
8.2.2 Atom “linter-harbour” ekstenzija.....	45
8.2.3 Atom editor “harbour-plus” ekstenzija.....	46
8.2.4 Atom editor osnovne funkcije.....	47
8.2.5 Zaključak.....	49
8.3 Eliminacija programskog “otpada”.....	50
8.3.1 Stanje 2014.....	50
8.3.2 Stanje 2018.....	51
8.3.3 Zaključak.....	51
8.4 “F18 update” - provjera i instalacija novih verzija.....	52
8.4.1 Zaključak.....	53
8.5 F18 automatski izvještaji o greškama (eng. bug reports).....	54
8.5.1 Serverski log.....	55
8.5.2 “Call stack” i stanje sistema u trenutku greške.....	56
8.5.3 Svrha i značaj automatskih “bug report”-a.....	57
8.5.4 Zaključak.....	57
8.6 F18 automatska integracija (CI).....	58
8.6.1 Zaključak.....	60
9 Iz osobne prakse.....	61
9.1 Software koji se ne koristi je otpad.....	61
10 Zaključak.....	62

IZJAVA O AUTORSTVU

Ja, **Husremović (Šekib) Ernad**, student Fakulteta informacijskih tehnologija, Univerziteta „Džemal Bijedić“ u Mostaru, pod punom moralnom, materijalnom i krivičnom odgovornošću, izjavljujem da je rad pod naslovom:

“Agilni razvoj softvera”

u potpunosti rezultat sopstvenog istraživanja, gdje su korišteni sadržaji drugih autora jasno označeni pozivanjem na izvor i ne narušavaju bilo čija vlasnička ili autorska prava.

U Sarajevu, 13.09.2018.

Ernad Husremović, LK 28110A536

Sažetak

Agilni razvoj softvera predstavlja pomak od striktno inžinjerske linearne i planirane paradigme prema adaptivnoj i manje formalnoj paradigmii baziranoj na ideji iskorištavanja resursa koji su na raspolaganju kako bi se klijentu isporučio **softver koji radi**. Može se reći da je agilni razvoj pomak od ideje da je samo savršeno dovoljno dobro ka ideji da proizvod ne mora biti savršen da bi bio dovoljno dobar i da uvijek postoji prostor za **usavršavanje**. U prvom dijelu rada biće prikazana osnovna ideja agilnog razvoja softvera kroz Manifest njegovih glavnih tvoraca, potom će se napraviti usporedba tradicionalnih metoda i metoda agilnog razvoja softvera, te na kraju prikaz najznačajnih metoda agilnog razvoja. U drugom dijelu rada će se prezentovati četverogodišnji ciklus razvoja realnog projekta, te analizirati realizaciju u kontekstu primjene koncepata agilnog razvoja.

Ključne riječi:

softver, agilni razvoj, upravljanje projektima, ekstremno programiranje (XP), scrum, kanban, lean, psihologija radnog mjesti, motivacija, visualization, atom programmerski editor, harbour programski jezik, xBase, DBF, PostgreSQL, upravljanje revizijama izvornog kôda (VCS), git, testiranje na prvom mjestu (TDD), kontinuirana integracija (CI), bring.out, F18 knjigovodstvo

Keywords:

software, agile development, project management, extreme programming (XP), scrum, kanban, lean, psychology at work, motivation, vizualizacija, atom programming editor, harbour programming language, xBase, DBF, PostgreSQL, version control system (VCS), git, test driven development (TDD), continous integration (CI), bring.out, F18 accounting

1 Uvod

Agilni razvoj softvera temelji se na Manifestu koji utvrđuje osnovne vrijednosti i principe. Na tim temeljima su nastale različite agilne metode koje unutar sebe objedinjuju određene prakse i koncepte. Agilni razvoj prepoznaje specifičnosti razvoja softvera u odnosu na druge inžinjerske grane. Razvoj softvera u gotovo svakom projektu obuhvata veliki dijapazon operacija od jednostavnih repetitivnih operacija koje nalikuju radu na tvorničkoj traci preko klasično inžinjerskih tehnika koje su slične gradnji kuće ili mosta do kreativnih procesa bliskih kreiranju umjetničkih djela.

1.1 Manifest agilnog razvoja softvera

Manifest agilnog razvoja softvera¹ je proglašen grupom 17 softver developerima objavljen 2001. godine:

“Mi otkrivamo bolje načine razvoja softvera, primjenjujemo ih i pomažemo drugima da ih primjenjuju proizvodimo softver i drugima pomažemo u njegovoj izradi. Kroz ovaj rad naučili smo cijeniti:

Osobe i njihove interakcije više od procesa i alata

Softver koji funkcioniše više od obimne dokumentacije

Saradnju sa klijentom više nego pregovore oko ugovora

Prilagođavanje promjenama više nego striktno slijedenje plana

Iako stavke na desnoj strani imaju svoju vrijednost, mi više cijenimo stavke na lijevoj strani.”

1.2 Osnovni principi u agilnom razvoju softvera

Tvorci manifesta navode principe na kojima se bazira razvoj softvera u novoj paradigmi:

“Slijedimo sljedeće principe:

1. *Naš najveći prioritet je zadovoljiti klijenta kroz rane i kontinuirane isporuke funkcionalnog softvera.*
2. *Prihvatamo promjene promjene u softverskim zahtjevima, čak i u ranoj fazi razvoja. Agilni procesi usvajaju promjene koje će obezbjediti klijentu kompetativnu prednost.*
3. *Stalno isporučujemo funkcionalan softver, u intervalima od nekoliko sedmica do par mjeseci, preferirajući kraće vremenske cikluse.*
4. *Poslovni ljudi i developeri trebaju kontinuirano, na dnevnoj osnovi, raditi zajedno na projektu.*
5. *Izradu projekata povjeriti motiviranim pojedincima. Obezbeđujemo im kvalitetno okruženje i svu potrebnu podršku, imamo povjerenje u njih da će posao biti završen kako treba.*
6. *Najefikasniji način prenosa informacija potrebnih za razvoj je direktna (lice u lice) komunikacija između učesnika.*

1 <http://agilemanifesto.org/>

7. Funkcionalan softver je primarna metrika napretka projekta.
8. Agilni procesi promoviraju održivi razvoj softvera. Investitori, developeri, korisnici trebaju moći nastaviti učešće u održavanju softvera neograničeno.
9. Kontinuirano pažnja na tehničku izvrsnost i dobar dizajn povećava agilnost.
10. Jednostavnost - sposobnost da se maksimalizira količina posla koji **nije** potrebno realizovati je od esencijalne važnosti.
11. Najbolje arhitekture, zahtjevi i dizajn kreiraju samoorganizovani timovi.
12. Tim u regularnim intervalima analizira svoje djelovanje, traži načine da bude što efektivniji, te u skladu sa tim nalazima podešava i prilagođava svoje djelovanje.”

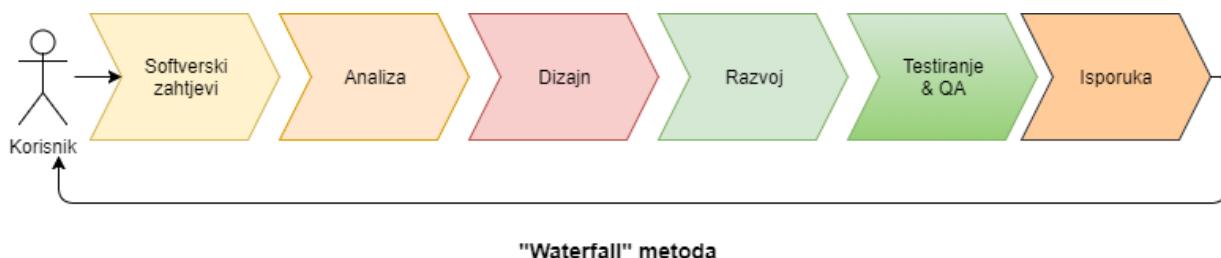
1.3 Usporedba tradicionalnih i agilnih metoda razvoja

1.3.1 Tradicionalne metode

Tradicionalne metode razvoja softvera slijede hijerarhijsku linearnu paradigmu koja odlikuje klasične inžinjerske projekte. Glavne karakteristike tradicionalnih metoda su:

- pojedinačne faze se obavljaju sekvencialno (analiza, dizajn, razvoj, testiranje, isporuka)
- pojedinačne faze (posebno u većim projektima) izvode različiti, izolovani timovi
- prije faze realizacije razrađuje se precizan i striktan vremenski plan koji je često integralni dio ugovora sa korisnikom
- Zahtjevi **definisani na početku** određuju vremenski plan, arhitekturu i realizaciju
- Obimna, ali **statična** dokumentacija
- U mnogim segmentima primjenjuju se metode koje su viđene u drugim inžinjerskim granama (npr. u građevinskim projektima)

Najpoznatiji metod koji se svrstava u tradicionalne metode razvoja softvera je “waterfall” model.



Na predhodnom dijagramu uočavamo dugačku povratnu petlju razvojnog tima - korisnik. Korisnik u ovom modelu ima priliku da pošalje povratnu informaciju o kvaliteti produkta tek nakon što su

okončane sve faze što projekta. Ova relativno kasna povratna informacija stvara velike izazove u modifikaciji početnog produkta jer korisnik gubi mogućnost da razumijeva produkt kroz sve faze, a to vodi ka tome da je njegova povratna informacija nespecifična i nefokusirana, pa često i nerazumljiva.

1.3.2 Agilne metode

Bazirajući se na osnovnim vrijednostima i principima agilnog razvoja softvera, razvilo se niz agilnih metoda koje unutar sebe obuhvataju određene elemente - prakse.

Najpoznatije metode su:

- Ekstremno programiranje (XP)
- “Scrum”
- “Lean software development”
- “Kanban”

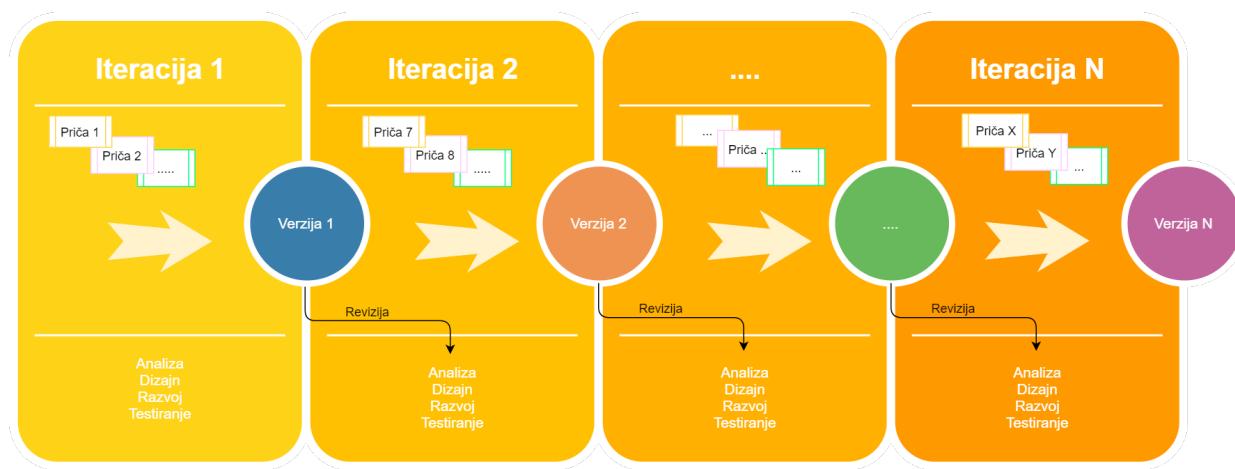
2 Ekstremno programiranje (XP)

2.1 Uvod

Prihvatanje promjena (eng. embracing changes) je osnovno polazište metode “Ekstremno programiranje” (XP). Dva ključna mesta unutar kojih se primjenjuje ovaj princip su granice projekta i programski kôd. Prakse i koncepti XP-a obezbeđuju da razvojni tim bude sposoban da prihvati i realizuje takve promjene na najbolji način. Zato npr. novi korisnički zahtjev ili značajne promjene izvornog kôda nisu vanredne, nego očekivane aktivnosti.

2.2 Organizacija XP tima

XP tim radi u sedmičnim iteracijama. Svake sedmice, tim obavlja analizu, dizajniranje, razvoj i testiranje. Osnovna jedinica rada je “priča” (eng. story). Priče predstavljaju funkcije ili dio funkcija softvera koje imaju određenu vrijednost za korisnika. Svake sedmice tim isporučuje 4-10 priča. Na kraju iteracije vrši se interna revizija softvera. Ukoliko je moguće, revizija se isporučuje krajnjim korisnicima na testiranje[AART].



Metoda "Ekstremno programiranje" (XP)

Tim radi u otvorenom prostoru koji omogućava brzu izmjenu informacija i ideja putem direktnе komunikacije (lice u lice). Ako postoji mogućnost, zajedno sa timom nalaze se predstavnici korisnika i koji praktično predstavljaju produženi dio tima. Oni učestvuju u konstrukciji priča, testiranju tokom razvoja, kao i revizijama softvera na kraju iteracija. Ukoliko krajnji korisnici nisu dostupni na licu mjesta, jedan od članova tima se imenuje za zastupnika korisnika (eng. proxy user), tako da se njemu postavljaju pitanja upućena korisnicima. Najčešće proizvod-menadžer (eng. product manager) preuzima ulogu proksija.

Određene prakse su posebno istaknute u XP metodologiji:

- Restruktuiranje programskog kôda (eng. refactoring)
- Programiranje u paru (eng. pair programming)
- Testiranje na prvom mjestu (eng. test driven development)

2.3 Restruktuiranje - refactoring programskog kôda

Razvoj softvera se realizira pisanjem izvornog kôda. Svaka funkcija softvera se može na više načina izraziti izvornim kôdom. Refactoring je proces izmjene strukture izvornog kôda **bez promjene** funkcionalnosti. Svrha refactoringa je učiniti kôd preglednijim, smanjiti mogućnost grešaka, optimizirati korištenje računarskih resursa ili učiniti izvođenje bržim.

2.4 Programiranje u paru

Programiranje u paru je tehnika u kojoj jedan developer sjedi za tastaturom (“vozač”, eng. driver), a drugi sjedi pored njega (“posmatrač”, eng. observer) i prati na ekraru njegov unos. U toku kucanja programskog kôda, svaki put kada se pojavi nejasnoća ili dilema, developeri započinju diskusiju. Primjenom ove tehnike postižu se sljedeći efekti:

- “posmatrač” na licu mjesta vrši pregled i kontrolu kvaliteta i čitljivosti programskog kôda (eng. code review)
- izvorni kôd je u startu zajednička a ne individualna vrijednost
- stalno se dešava intenzivna i kvalitetna komunikacija unutar razvojnog tima
- članovi razvojnog tima dijele i prenose znanje unutar tima (eng. sharing knowledge)

2.5 Testiranje na prvom mjestu (TDD)

Razvoj vođen testiranjem (eng. Test Driven Development) je praksa u pisanju softvera kojom se pisanje započinje pisanjem testnih funkcija. Na taj način izvorni kôd je u potpunosti pokriven setom testova. Testovi imaju sljedeće funkcije:

- Testovi dokumentuju namjere izvornog kôda na najbolji način
- Izrada testnih funkcija je proces koji na najboljli način kombinuje razvoj i dizajn. Prije implementacije developer može vidjeti kako će kôd biti korišten
- Refactoring testabilnog kôda je efikasan i jednostavan, sa minimalnim rizicima

2.5.1 Rust TDD i refactoring primjer

U ovom primjeru demonstriraćemo TDD i refaktoring na primjeru rješavanja problema projekta Euler² sa rust programskim jezikom³.

[Priča 1]

Euler problem 1:

- Naći sumu svih prirodnih brojeva koji su djeljivi bez ostatka sa 3 i 5, manji od zadatog broja N.
- Ako se za graničnu vrijednost postavi 10, rezultat je 23.

² <https://projecteuler.net/problem=1>

³ <https://hackernoon.com/a-taste-of-rust-6d8fc60e050> ; https://github.com/hernad/rust_tdd

Opis problema predstavlja ujedno prvi korisničku priču našeg programa. Iz nje direktno možemo definisati i prvu testnu funkciju:

```
##[test]
fn euler_problem_1_test_1() {
    assert_eq!(euler_problem_1(10), 23 );
}
```

Da bi se program mogao kompajlirati, dodajemo i praznu funkciju (skeleton) :

```
fn euler_problem_1(max: u64) -> u64 {
    unimplemented!();
}
```

Po prvi put pokrećemo test, sa očekivanom rezultatom - neuspješno:

```
$ cargo test
  Finished dev [unoptimized + debuginfo] target(s) in 0.01s
    Running target\debug\deps\rust_tdd-0825df093aadf3ef.exe

running 1 test
test euler_problem_1_test_1 ... FAILED

failures:

---- euler_problem_1_test_1 stdout ----
thread 'euler_problem_1_test_1' panicked at 'not yet implemented', src\main.rs:7:5
note: Run with `RUST_BACKTRACE=1` for a backtrace.

failures:
    euler_problem_1_test_1

test result: FAILED. 0 passed; 1 failed; 0 ignored; 0 measured; 0 filtered out
error: test failed, to rerun pass '--bin rust_tdd'
```

Na kraju testa dobijamo rezultat "crvene" boje. Kada se uspješno realizuje implementacija funkcija koje testiramo, test "pozeleni". Jednostavni tok razvoja tokom koga programer traži rješenje u kome će svi testovi postati zelene boje je standardna rutina u TDD procesu. Jednostavnost, vizuelna reprezentacija (crveno/zeleno) su veoma značajne za efikasnost i kvalitet procesa. Developer postiže maksimalni **fokus** i produktivnost u razvojnog ciklusu. Programiramo prvo rješenje:

```
fn euler_problem_1(_max: u64) -> u64 {
    let mut result = 1;
    let mut i = 0;
    loop {
        if i >= _max {
            break;
        }
        if i % 3 == 0 || i % 5 == 0 {
            result += i;
        }
        i += 1;
    }
    result
}
```

Testiramo ga:

```
$ cargo test  
running 1 test  
test euler_problem_1_test_1 ... FAILED  
  
failures:  
---- euler_problem_1_test_1 stdout ----  
thread 'euler_problem_1_test_1' panicked at 'assertion failed: `(left == right)`  
  left: `24`,  
  right: `23`', src\main.rs:3:5  
note: Run with `RUST_BACKTRACE=1` for a backtrace.  
  
failures:  
  euler_problem_1_test_1  
  
test result: FAILED. 0 passed; 1 failed; 0 ignored; 0 measured; 0 filtered out  
error: test failed, to rerun pass '--bin rust_tdd'
```

Rezultat provedenog procesa je negativan. Izvještaj testiranja kaže da je lijeva strana testa dala rezultat 24, umjesto željenog 23. To nas brzo dovodi do pogrešne inicijalne vrijednosti varijable "result".

Nakon ispravke linije:

```
let mut result = 0;
```

Ponovo pokrećemo test, po prvi put uspješno:

```
running 1 test  
test euler_problem_1_test_1 ... ok  
  
test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```

Ovakav način razvoja (eng. development workflow) sadrži bitne motivacijske elemente: uspješnost testa stvara osjećaj postignuća kod developera, te daje dodatno motivaciju da se nastavi na podizanju kvalitete programskog kôda.

Željeni rezultat je postignut, ali je funkcija komplikovana. Uočavamo da nam je dovoljna jedna "for" petlja:

```
fn euler_problem_1( _max: u64 ) -> u64 {  
  
    let mut result = 0;  
    for i in 0.._max {  
        if i % 3 == 0 || i % 5 == 0 {  
            result += i;  
        }  
    }  
    result;  
}
```

Test kaže da je ova iteracija ispravna, znači da je naš prvi refactoring uspješan! Međutim, postojeći kôd je napisan proceduralno. Rust nudi napredne funkcionalne koncepte:

```
fn euler_problem_1( _max: u64 ) -> u64 {  
    let mut result = 0;  
    for i in (0.._max).filter(|n| n % 3 == 0 || n % 5 == 0) {  
        result += i;  
    }  
    result  
}
```

Pošto je `(0 .. _max)` rang koji implementira “Iterator” interfejs (jezikom “rust”-a “trait”; bos. osobina), na njega se mogu primjeniti iteratorske funkcije koje za argumente uzimaju anonimne funkcije (eng. closure). Iskoristićemo funkciju “filter”, čiji je argument anonimna funkcija `|n| n % 3 == 0 || n % 5 == 0` koja ispituje djeljivost argumenta. Ponavljam test, rezultat je i dalje “zelen”, što nam kaže da smo na pravom putu. Varijable koje mijenjaju svoju vrijednost (mutirajuće variable, “rust” zahtjeva da se te varijable eksplisitno označavaju sa rezervisanom riječi “mut”) nisu poželjne kada želimo primjenjivati funkcionalne koncepte programiranja. Da li nam uopšte treba varijabla “result”? Funkcionalno programiranje bazira se na paradigmi formiranja lanca funkcionalnih transformacija, po uzoru na matematičke transformacije “čistih” funkcija (funkcije koje nemaju sporedna dejstva; eng. “side effects”):

```
f1( arg ) -> f2(resultat_f1)-> ... > fn( resultat_fn-1 ) -> resultat_n
```

Ovaj koncept realiziramo u našem primjeru dodavanjem `sum()` funkcije:

```
fn euler_problem_1( _max: u64 ) -> u64 {  
    (0.._max).filter(|n| n % 3 == 0 || n % 5 == 0).sum()  
}
```

Varijabla “result” je izbačena. U skladu sa uobičajenom rutinom, pokrećemo test:

```
running 1 test  
test euler_problem_1_test_1 ... ok  
  
test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```

U nekoliko iteracija došli smo do kvalitetnog rješenja. U tom procesu naš stalni pomoćnik bila je je baterija testova.

Obradimo još jednu karakterističnu situaciju unutar XP tima. Razvojnom timu je pridružen član koji zastupa klijenta. On je poslovni analitičar. Njegov glavni alat je aplikacija za tabelarne proračune.

Dolazi nam sa proračunom Eulerove funkcije sa graničnim elementom 21. Traži od nas da provjerimo da li naša implementacija ispravno proračunava i taj slučaj:

A	B
1	1
2	0
3	3
4	0
5	5
6	6
7	0
8	0
9	9
10	10
11	0
12	12
13	0
14	0
15	15
16	0
17	0
18	18
19	0
20	20
21	98
22	
23	

Poračun "Eulerove funkcije" u "libreoffice calc"

Pravimo novu testnu funkciju:

```
#[test]
fn euler_problem_1_test_2() {
    assert_eq!(euler_problem_1(21), 98 );
}
```

Pokrećemo test:

```
$ cargo test
Compiling rust_tdd v0.1.0 (file:///D:/devel/rust_tdd)
Finished dev [unoptimized + debuginfo] target(s) in 0.83s
Running target\debug\deps\rust_tdd-0825df093aadf3ef.exe

running 2 tests
test euler_problem_1_test_1 ... ok
test euler_problem_1_test_2 ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```

Oba testa "ok", "zeleni" smo i dalje, posao je završen.

3 “Scrum”

3.1 Uvod

“Scrum” je termin u ragbiju koji se koristi za formaciju igrača kojom se započinje igra nakon prekida, u našoj terminologiji poznat kao “bula”.



Ragbi igrači u "buli" [wikipedia]

Osnovna paradigma koja definiše ovu metodu razvoja je prihvatanje principa neizvjesnosti, odnosno nemogućnosti razumjevanja svakog detalja problema i potpuno tačnog predviđanja posljedica u toku procesa. Stoga se “scrum” fokusira na maksimiziranje timskog potencijala da efektivno i efikasno izvršava zadatke, fleksibilno pristupa zahtjevima koji se pojavljuju tokom procesa, te se prilagođava razvoju tehnologije i zahtjevima tržišta.

3.2 Dnevni “Stand-up” sastanci (eng. stand-up meetings)

Dnevni “Stand-up” sastanci su efektivan način da se članovi tima međusobno informišu o stanju projekta. Kao što ime sugeriše, učesnici svoje izlaganje izlažu stojeći. Taj način izlaganja pomaže da sastanci budu kratki. Svakom učesniku se daje kratak period vremena za izlaganje.[APRAT]

Da bi sastanak imao potreban fokus, svaki učesnik odgovara na tri pitanja:

- Šta sam postigao juče?
- Šta planiram za danas?
- Koji su moje glavne prepreke i problemi da realiziram planirano?

3.3 “Sprintovi”

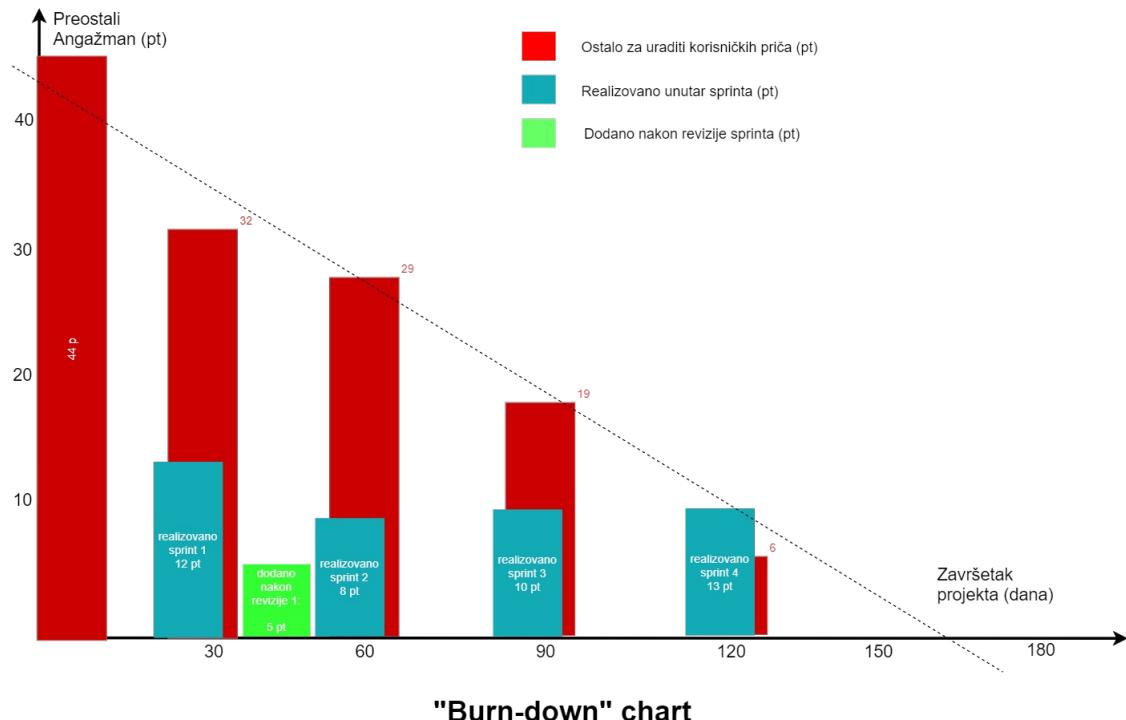
Unutar XP-a smo se upoznali sa konceptom iteracije kao jedinicom razvoja softvera. “Scrum” svoje iteracije naziva “sprintovi”. Pojedinačni sprint traje 15-30 dana. Vremensko ograničenje sprinta (eng. timeboxing) je obavezna praksa. Ako desi da planirane funkcije nije moguće završiti unutar sprinta, određuju se one koje se izbacuju za naredni sprint da bi se ispoštovalo vremensko ograničenje.

Kontinuirana vremenska dinamika, striktnost u trajanju sprintova vođi tima (Scrum masteru) omogućava da nakon nekoliko sprintova pouzdano utvrdi **kapacitet** tima (eng. team velocity). Ta metrika je ključna informacija za planiranje narednih sprintova, i naravno završetak cijelog projekta.

3.3.1 Centralna lista korisničkih priča (eng. Product backlog)

Centralna lista korisničkih priča (eng. Product backlog) je kolekcija priča (funkcija) koje korisnik želi imati u svom softveru.

Ona je **prioritizirana** od strane **korisnika**, te **procijenjena** (u smislu napora - potrebnog angažmana, eng. effort) od strane **razvojnog tima**.[ASAM]



3.3.2 Lista priča novog sprinta (eng. Spring backlog)

Iz centralne liste se na osnovu prioriteta i procjena angažmana formira lista priča (eng. Sprint backlog) koje će se realizovati unutar narednog sprinta. Lista za naredni sprint se utvrđuje nakon revizije neposredno završenog sprinta. Nakon revizije “Scrum master” ima svježe informacije o kapacitetu tima, te povratne informacije od korisnika. Te informacije mu pomažu da formira listu zadataka za realizaciju koju je tim **sposoban** realizovati, a korisnik najviše treba i očekuje.

4 “Lean software development”

4.1 Uvod

Ova metoda razvoja softvera se temelji na “Lean” metodama u oblasti proizvodnje, logistike i administrativnih operacija unutar finansijskih i trgovinskih organizacija⁴.

Doslovni prevod riječi “lean” sa engleskog znači mršav, vitak - bez viška kilograma. Termin asocira na glavni cilj ove metode: isporučiti korisniku vrijednost što brže eliminisanjem beskorisnog otpada i podizanjem kvalitete usluga i/ili proizvoda. “Lean” metodologija je bazirana na sedam praksi:



4.2 Eliminacija otpada

Otpad je sve što ne doprinosi vrijednosti finalnog proizvoda:

- Nepotrebna ili zastarjela dokumentacija
- Funkcije softvera koje niko ne koristi
- Neefikasne ili nepotrebne operacije i procesi

4.3 Kvalitet izrade

Kvalitetnom izradom softverskog proizvoda preveniraju se sofverske greške, te eliminišu operacije potrebne da se greške otklone nakon stavljanja u upotrebu.

⁴ <https://dzone.com/refcardz/getting-started-lean-software>

4.3.1 Kreiranje znanja

Saznanja stečena u prethodnim operacijama su baza znanja za naredne razvojne cikluse. To znanje postaje “gorivo” - akcelerator razvojnog tima.

4.3.2 Donošenje odluka u najboljem trenutku

Znanje se stiče u svakoj fazi rada. Što vrijeme odmiče, tim posjeduje više informacija i znanja za pravilno odlučivanje. Zato strateške odluke (npr. arhitektura rješenja u narednoj iteraciji) treba donijeti u najboljem trenutku - što je moguće kasnije.

4.3.3 Brze isporuke

Kada se dođe do određenog rješenja, ne treba odlagati sa njegovom isporukom. Rješenje koje je aktuelno u jednom trenutku, nakon određenog perioda može izgubiti na vrijednosti. Dodatno, u periodu neisporuke (bez opravdanog razloga) razvojni tim ne može dobiti povratne informacije od korisnika. Nedostatak informacija onemogućava rast znanja o proizvodu na kome se radi.

4.3.4 Uvažavanje ljudi

Ljudi su ti koji obezbeđuju razvoj (developeri) i ocjenjuju vrijednost softvera koji se za njih pravi (korisnici). Zato su komunikacija bazirana na uvažavanju i stvaranje dobrih uslova rada za razvojni tim obavezni elementi projekta organiziranog po “Lean” principima.

4.3.5 Optimiziranje cjeline

Posmatranje projekta u cijelini (eng. “see the whole”, “big picture”) je princip djelovanja i razmišljanja o projektu koji garantuje da sve aktivnosti koje se poduzimaju vode ka najboljem mogućem rezultatu. Ova praksa pomaže da sve do sada navedene prakse rezultiraju sinergijskim učinkom.

5 “Kanban”

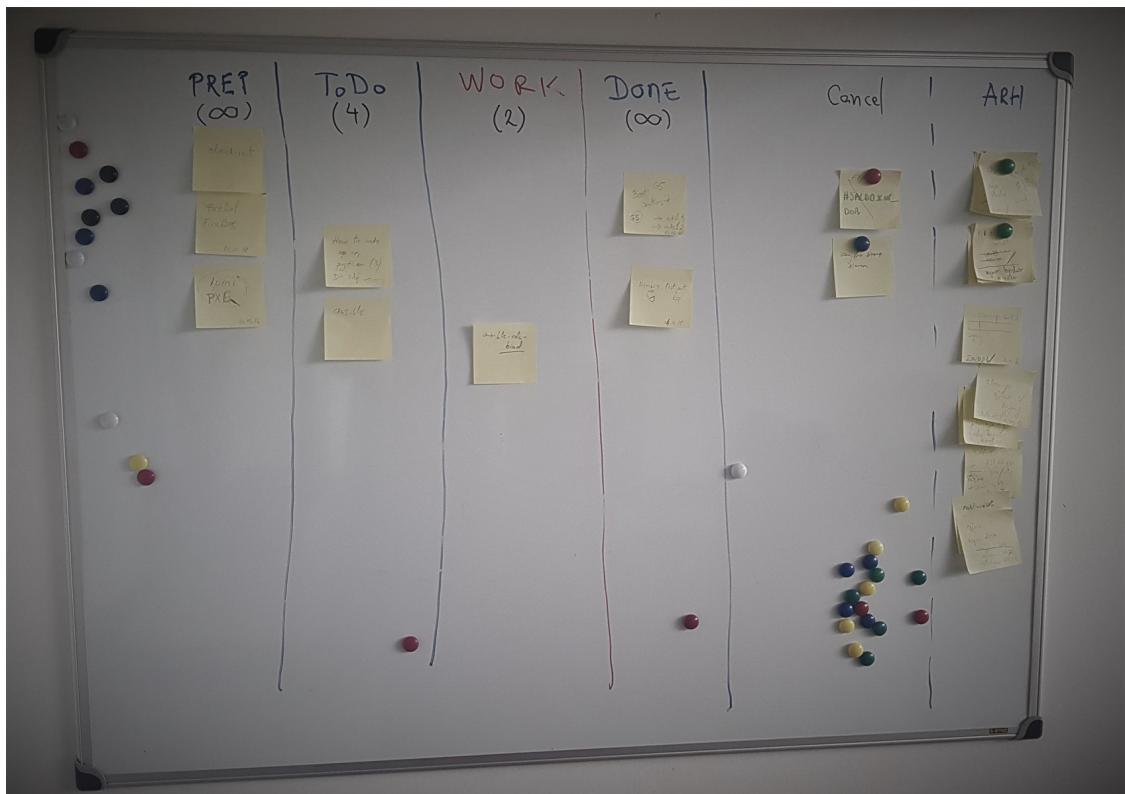
5.1 Uvod

“Kanban” u prevodu sa japanskog znači “pano”, “oglasna tabla”. “Kanban” je sistem za planiranje proizvodnje kao podrška za “lean manufacturing” i “just-in-time manufacturing”. Inžinjer u “Toyoti” Taiichi Ohno je razvio kanban metodu da bi povećao efikasnost proizvodnje⁵.

U oblasti razvoja softvera “Kanban” pripada grupi “Lean” metoda, u kojoj je “kanban” ploča centralno mjesto vizuelnog prikaza stanja i planiranja toka operacija. “Kanban” je veoma podesan u poslovima podrške korisnicima.

“Kanban” omogućava timu da bolje razumije operacije u toku. Limitiranje broja operacija obezbjeđuje redukciju otpada nastalog uporednim obavljanjem više operacija uslijed promjene konteksta (eng. multitasking and context switching waste)⁶. Limitiranjem se takođe obezbjeđuje fokus na operacije koje će otkočiti dovođenje operacija do krajnjeg cilja.

5.2 “Kanban” ploča



Kanban ploča u preduzeću "bring.out"

5 <https://en.wikipedia.org/wiki/Kanban>

6 [https://en.wikipedia.org/wiki/Kanban_\(development\)](https://en.wikipedia.org/wiki/Kanban_(development))

Na slici je prikazana "kanban" ploča koja se koristi za poslove podrške korisnicima. Ono što je vrlo važno uočiti jeste da svaki tim organizuje kanban ploču u skladu sa svojim načinom rješavanja zadataka. Gornja ploča sadrži sljedeće kolone:

Oznaka kolone	Opis	Limit	Objašnjenje
PREP	Priprema (eng. preparation)	∞	Pripremne radnje potrebne da bi se moglo pristupiti rješavanju
ToDo	U redu čekanja (eng. to do)	4	Zadaci iz pripreme se stavljuju u skladu sa prioritetima u "ToDo" listu. Ova lista je ograničena na 4 stavke. Razlog preglednost, kao i činjenica da je naredna, radna kolona limitirana na dvije stavke
Work	U realizaciji	2	Zadaci u realizaciji. Limit od dvije stavke određen je trenutnim kapacitetom tima.
Done	Realizovano	∞	Zadaci koji su realizovani
Cancel	Otkazano	-	Otkazani zadaci se drže u ovoj koloni ograničeni period, dok postoji mogućnost da se zadatak vrati (u istoj ili promjenjenoj formi) u rad
Arh	Arhiva	-	Arhiva realizovanih aktivnosti. Na kraju određenog perioda, u našem slučaju na godišnjem nivou se "čisti" - prazni.

Organizacija "kanban" ploče se može mijenjati u skladu sa potrebama i stanjem tima. Npr. u slučaju da se kapacitet tima poveća, limit na "ToDo" i "Work" kolonama se može povećati. Ukoliko se pokaže da ploča nije dovoljno informativna, može se dodati nova kolona ili izbrisati postojeća.

Kartice na ploči očekivano predstavljaju zadatke. Na karticama navode osnovne informacije koje tim koristi u realizaciji:

- Opis zadatka
- Datum zaprimanja zadatka
- Procjena potrebnog angažmana (u satima ili tačkama)
- Referentni broj u aplikaciji za praćenje korisničkih zahtjeva (eng. bug. tracker)
- Datum realizacije

Kartica se postupno ažurira potrebnim informacijama. Kartice se mogu označavati različitim bojama u skladu sa njihovim tretmanom u procesu rješavanja problema, npr.:

- Crvena - kritične greške koje onemogućavaju rad korisnika
- Zelena - novi zahtjevi
- Narandžasta - prilagodbe korisničkog interfejsa
- Plava - rad na izvještajnim opcijama aplikacije

Ono što je najbitnije uočiti jeste da su organizacija kanban ploče i oznake na karticama kontekstualizirane načinom na koji tim koji rješava svoje zadatke. Ono što je za jedan tim bitno, za drugi može biti prenatrpano informacijama. Informativnost i preglednost koje obezbjeđuje vizuelna reprezentacija, te jednostavno ažuriranje informacija su ključni atributi ove metode razvoja.

“Kanban” ploča može biti virtuelna ili fizička. Fizička ploča je podesna zato što je ona odličan medij za diskusije tima o poslovima koje treba obaviti. Kao takva, ona je idealno mjesto za organizovanje “stand-up” sastanaka. U slučaju da su članovi tima dislocirani, ili je kanban aplikacija već integrirana sa ostalim alatima koje tim svakodnevno koristi, umjesto fizičkog koristi se virtuelna “kanban” ploča. Primjer takve aplikacije je waffle.io⁷:

The screenshot shows a Kanban board titled "waffleio/serenity". The board is divided into four columns: "Ready" (5 items), "In Progress" (4 items), "Needs Review" (1 item), and "Done" (4 items). Each column has a header with a "Move" icon and a "Count" button. Below each header is a list of tasks with their IDs, descriptions, labels, and user icons. The "Ready" column has tasks: 20 disable explosive set by trap (expedite), 18 recover hidden loot at Canton (financial), 4 retrieve cargo from train (train job, enhancement), 30 join Mal in boarding train (train job), and 21 collect remaining funds to pay for shipmates release (financial). The "In Progress" column has tasks: 1 alert others of distress call (expedite), 6 fix ship's engine problem (bug, blocked), 13 unload and pen cattle (help wanted), and 2 get cargo from abandoned carrier. The "Needs Review" column has task: 64 prepare for duel. The "Done" column has tasks: 29 find a brand new compression coil for the steamer (wontfix), 5 find a captain for the ship (startup), 27 find a mechanic for the ship (startup), and 16 buy a solid ship (startup).

prikaz ekrana “waffle.io” aplikacije [github.com]

⁷ <https://github.com/waffleio/waffle.io>

6 Zajedničke prakse i koncepti

Svaka od navedenih metoda ima fokus na određene prakse i koncepte. Međutim, određene prakse su postale sastavni dio svake savremene metode razvoja softvera.

6.1 Upravljanje revizijama izvornog kôda (VCS)

Sa softverom za upravljanje verzijama izvornog kôda (eng. version control system - VCS) članovi razvojnog tima imaju uvid u istoriju promjena na softveru. Izvorni kôd, kao glavni artifakt softvera, ima posebno mjesto u razvojnom ciklusu. Gotovo sve aktivnosti u razvoju softvera u konačnici se završavaju pisanjem novog ili ispravkom postojećeg kôda.

“Softver za kontrolu verzija izvornog kôda je sistem koji bilježi promjene na datoteci ili setu datoteka tokom vremena tako da se specifične verzije mogu naknadno vratiti.” [PROGIT]

Baza promjena izvornog kôda je osnovni alat svakog developera. Prvi ovakvi alati su bili organizovani na klijent-server principu (CVS⁸, subversion⁹). Kod tih alata na lokalnoj strani (računaru developera) postoji samo jedna revizija - aktuelna revizija na kojoj radi. Za pristup ostalim revizijama mora se kontaktirati VCS server. Danas međutim dominiraju distribuirani VCS alati (Git¹⁰, Mercurial¹¹). Iako “Mercurial” koristi nekoliko velikih kompanija i projekata (Facebook, Mozilla, openjdk), “Git” je danas najviše zastupljen. Popularizaciji “Git”-a su značajno pomogli web servisi “Github”¹² i “Gitlab”¹³. Ovi servisi su mnogo više od VCS alata. Oni sadrže komponente koje obuhvataju kompletan razvojni ciklus softvera:

- Softver za kontrolu verzija (VCS)
- Generacija niza statistika vezanih za izvorni kôd
- Organizovanje razvojnog tima
- Praćenje grešaka i zahtjeva (eng. bug tracking)
- Automatska integracija binarnih verzija (eng. continuous integration)
- Dokumentacija softvera (wiki stranice)

Kod distribuiranog VCS softvera svaki developer ima sopstvenu kopiju distribuirane baze. On može da radi na svojoj kopiji bez pristupa centralnom serveru. Zato distribuirani način rada zahtjeva rješavanje kolizija prilikom spajanja verzija više developera koji su radili na istom kôdu (eng. merging and resolving conflict). Kod prvih verzija distribuiranih VCS alata ti algoritmi nisu bili usavršeni, pa su dominirala klijent-server rješenja. Međutim, Linus Torvalds je 2005, kao vođa Linux projekta izdao prvu verziju “git”-a. Linus je do tada za razvoj “Linux”-a koristio jedan komercijalni distribuirani VCS alat. Kada je dalje korištenje tog alata bilo nemoguće radi licencnih restrikcija, Linus nije prihvatio prijedlog da se razvoj Linuxa odabere neki centralizirani “opensource” VCS alat (što je u to vrijeme bila jedina opcija ako se ograniči na otvoreni softver), nego je odlučio napraviti sopstveni distribuirani VCS alat. U narednim godinama oko “Git”-a je razvijen čitav eko-sistem servisa i aplikacija.

8 https://en.wikipedia.org/wiki/Concurrent_Versions_System

9 https://en.wikipedia.org/wiki/Apache_Subversion

10 <https://git-scm.com>

11 <https://www.mercurial-scm.org>

12 <https://github.com>

13 <https://gitlab.com>

- Komunikacija unutar razvojnog tima i sa korisnicima integracijom “chat” servisa (razgovor u realnom vremenu)
- Pregled izvornog kôda (eng. code review workflow)

```

M/d-devel/F18_knowhow
F18_os.cloc
F18_pos.cloc
F18_scripts.cloc
F18_virm.cloc
template/
nothing added to commit but untracked files present

hernad@DESKTOP-OHRJEDF MINGW64 /d-devel/F18_knowhow
$ git commit -am "copyright 2018"
[3-std a4e8fe4] copyright 2018
 1 file changed, 2 insertions(+), 2 deletions(-)

hernad@DESKTOP-OHRJEDF MINGW64 /d-devel/F18_knowhow
$ git commit -am "BUILD_RELEASE 3.1.215"
[3-std 231bd1fd] BUILD_RELEASE 3.1.215
 1 file changed, 1 insertion(+), 1 deletion(-)

hernad@DESKTOP-OHRJEDF MINGW64 /d-devel/F18_knowhow
$ git tag 3.1.215

hernad@DESKTOP-OHRJEDF MINGW64 /d-devel/F18_knowhow
$ git push origin 3-std --tags
Enumerating objects: 1323, done.
Counting objects: 100% (1323/1323), done.
Delta compression using up to 8 threads.
Compressing objects: 100% (701/701), done.
Writing objects: 100% (703/703), 279.25 KiB | 4.58 MiB/s, done.
Total 703 (delta 622), reused 0 (delta 0)
remote: Resolving deltas: 100% (622/622), completed with 603 local objects.
To github.com:knowhow/F18_knowhow.git
  d637d580..231bd1fd 3-std -> 3-std
 * [new tag]           3.1.215 -> 3.1.215

hernad@DESKTOP-OHRJEDF MINGW64 /d-devel/F18_knowhow
$ |

```

Osnovne git operacije u konzolnom režimu - commit, tag, push

```

$ git log -n 10
commit 91af1aef8622d9f785a7197fc5e6e2675c7cdd9 (HEAD -> 3-std)
Author: Ernad Husremovic <hernad@bring.out.ba>
Date:   Mon Aug 20 15:40:58 2018 +0200

    F18 nalog count out

    commit a689788b1577e463adb08ff854cfcc25c78bceea (origin/HEAD, origin/3-std)
    Author: Ernad Husremovic <hernad@bring.out.ba>
    Date:   Mon Aug 20 15:38:51 2018 +0200

        unzbrka

    commit 33b9be3ea73f5f72587d077a5e756283d5bbb154
    Author: Ernad Husremovic <hernad@bring.out.ba>
    Date:   Mon Aug 13 18:27:05 2018 +0200

    ....
    ....

```

git log promjena, pregled u konzolnom režimu

6.1.1 Github repozitorij F18 projekta

F18 - simple accounting for Bosnians <http://redmine.bring.out.ba/projects/f18>

Author	Commit Message	Date
jdone	2i ld_radsht hack	3 years ago
cloc	unzbrka	3 days ago
common	unzbrka	3 days ago
common_legacy	Copyright 2018	10 days ago
core	Copyright 2018	10 days ago
core_dbf	Copyright 2018	10 days ago
core_pdf	pdf SumatraPDF	a year ago
core_reporting	Copyright 2018	10 days ago
core_senafori	Copyright 2018	10 days ago
core_sql	Copyright 2018	10 days ago
core_string	Copyright 2018	10 days ago
core_ui2	Copyright 2018	10 days ago
darwin	darwin back na gbtwc	4 years ago
doc	kalk razdruz_magacin_na_osnovu_pos_prodrage	10 months ago
epdv	Copyright 2018	10 days ago

Github: Pregled datoteka

Default branch

Branch	Last Update	Status
3-std	Updated 3 days ago by hernad	Default

Your branches

Branch	Last Update	Status	Size	Action
3	Updated 5 months ago by hernad	green	379 0	New pull request
3-vindi	Updated 9 months ago by hernad	yellow	490 45	New pull request
23100-1d	Updated a year ago by hernad	red	851 3	New pull request
hbcurl	Updated a year ago by hernad	green	1341 1	New pull request
23100-fakt	Updated a year ago by hernad	yellow	1471 55	New pull request

Stale branches

Branch	Last Update	Status	Size	Action
semaphores	Updated 5 years ago by hernad	red	4618 0	New pull request
1.4	Updated 4 years ago by hernad	yellow	4754 3	New pull request
ng2	Updated 3 years ago by bringout	red	4979 125	New pull request
1.7	Updated 3 years ago by hernad	green	2959 1	New pull request
23100-fakt	Updated a year ago by hernad	yellow	1471 55	New pull request

Github: Pregled "branch"-ova projekta

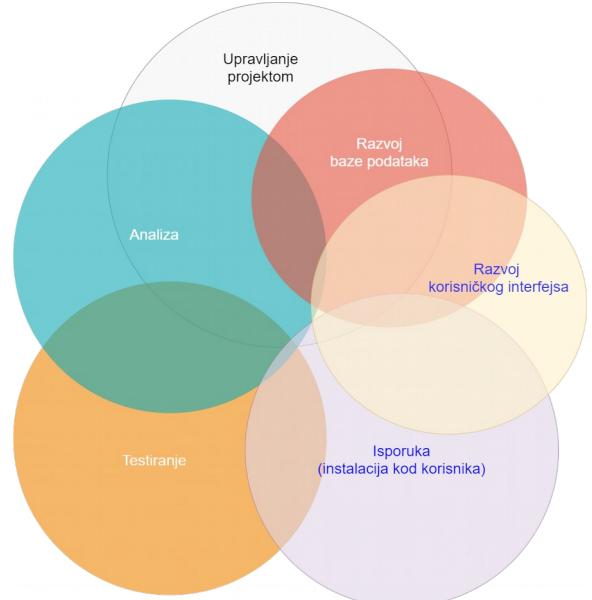
The screenshot shows a GitHub repository named 'knowhow / F18_knowhow'. The 'Code' tab is selected. A dropdown menu indicates the branch is '3-std'. The commit history is displayed in two sections: 'Commits on Aug 20, 2018' and 'Commits on Aug 13, 2018'. Each section contains several commits by the user 'unzbrka' (hernad). Commit messages include 'F18 cloc 1.7 branch 1.7.56 - 04/2014', 'cloc 3.1.215', 'cloc cleanup', 'BUILD_RELEASE 3.1.215', 'Copyright 2018', 'Copyright 2018', 'os refactor: out nepotreban komentar fja, fix vars', and 'fix LOCAL i'. Each commit has a detailed view icon, a copy icon, and a share icon.

Github: Pregled revizija (eng. commits)

6.2 Multi-funkcionalni tim

Agilne projekte karakteriše činjenica da uloge pojedinih članova nisu striktne. Članovi tima uzimaju svaki zadatak koji će doprinijeti projektu - bez obzira na titulu i glavnu ulogu u projektu[ASAM].

Naravno, odabir zadataka je primarno baziran na ekspertizi, glavnoj ulozi, preferencijama i efikasnosti članova. Međutim, ako primjera radi određeni zadatak traži angažman cijelog tima da bi se postigao projektni cilj, svi se angažuju na rješavanju tog zadataka.



Uloge pojedinih članova tima postoje, ali nisu striktne

6.3 Agilni trener

Agilni trener pomaže timu da se usvoje agilne prakse. On ili ona pomaže timu da usvoji agilni pristup rješavanju zadataka, te da pomogne otklanjanju mentalnih, emocionalnih i tehničkih barijera koje onemogućavaju tim da djeluje agilno[LEARNAG].



Trener američkog fudbala (dječja liga) daje upute članovima tima [pixabay.com]

Agilni trener radi sa svakim članom tima na način da mu objašnjava ne samo kako da primjeni određene agilne prakse, nego i da razumije **zašto** to radi. Postoji niz primjera timova koji su nisu postigli nikakav beznačajan napredak primjenom agilnih metoda upravo zato što nisu promijenili način razmišljanja (eng. mindset). Multi-funkcionalni tim, iterativni pristup, direktnu komunikaciju lice-u-lice, promjene u hijerarhijskoj strukturi tima daće pozitivne efekte samo pod uslovom da članovi tima razumiju zašto funkcionišu na taj način. Razumjeti “zašto” je posebno bitno u organizacijama koje imaju dugogodišnje “tradicionalne” strukture i organizacije.

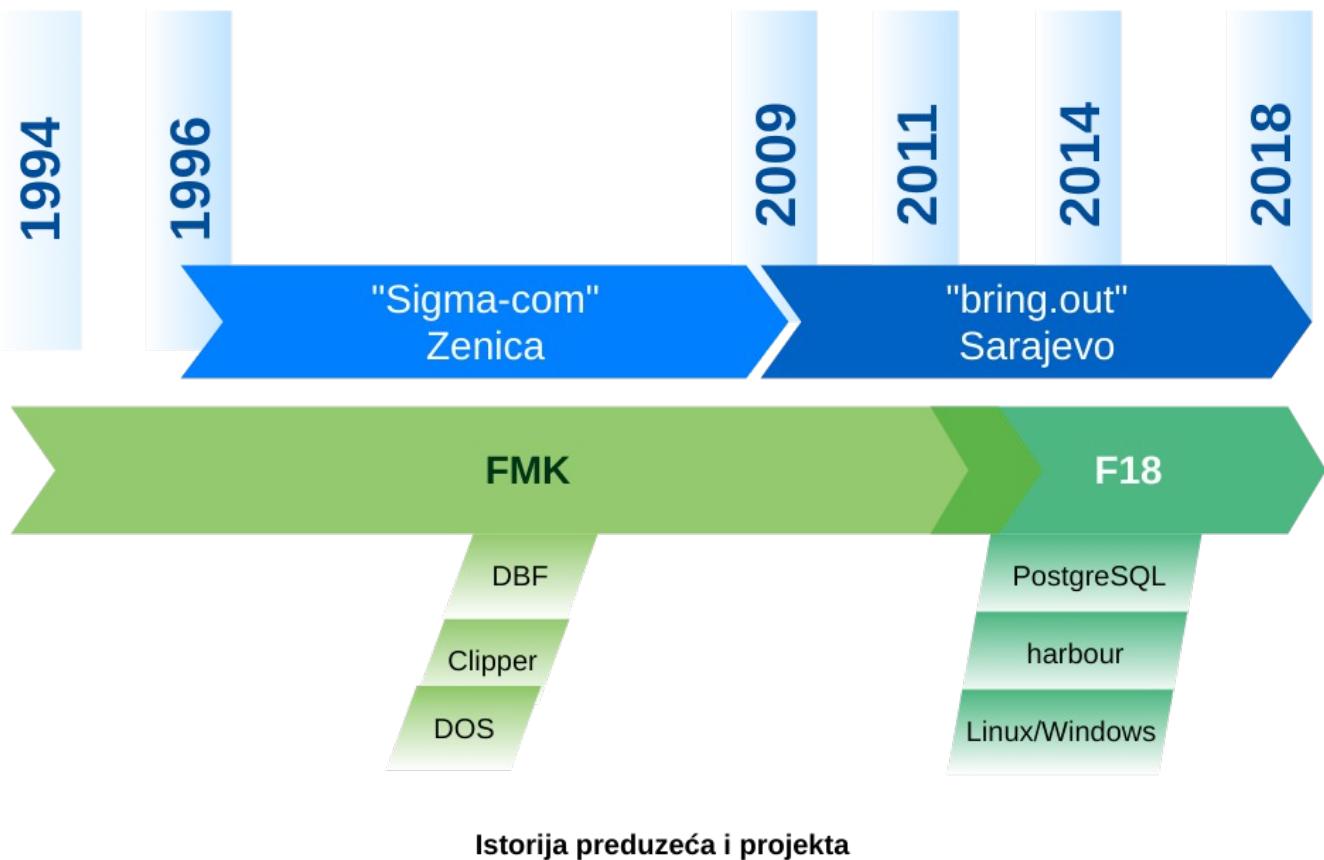
Svaka organizacija je određena kontekstom unutar koga djeluje. Taj kontekst kreiraju interne vrijednosti i odnosi unutar tima, korisnici, te pravila industrije za koju se problem rješava.

Agilni trener je sposoban da u konkretnom kontekstu primjeni odgovarajuću agilnu metodu. On dobro razumije agilne vrijednosti i principe na osnovu sopstvenih i iskustava organizacije drugih timova.

7 F18 - “knjigovodstvo za Bosance”

7.1 Uvod

F18 je knjigovodstvena aplikacija¹⁴ preduzeća “bring.out” namjenjena za bosansko-hercegovačko tržište. Projekat je započet 2011 godine, a prve verzije objavljene 2012 godine. Aplikativni kôd međutim nije pisan ispočetka, nego je preuzet do “FMK”¹⁵ projekta istog preduzeća. Prva verzija “FMK” je objavljena 1994 godine.



7.1.1 Cilj projekta

Polazna osnova F18 bila je izrada aplikacije koja će u funkcionalnom smislu biti identična “FMK”, pri čemu će eliminisati tehnološka ograničenja svog prethodnika. “FMK” je imao sljedeće nedostatke:

- Aplikacija pisana u napuštenom programskom jeziku “Clipper”¹⁶, koji je pisan za DOS operativni sistem¹⁷.

14 https://github.com/knowhow/F18_knowhow

15 <https://github.com/bringout-fmk>

16 [https://en.wikipedia.org/wiki/Clipper_\(programming_language\)](https://en.wikipedia.org/wiki/Clipper_(programming_language))

17 <https://en.wikipedia.org/wiki/DOS> ; “FMK” je radio pod “Windows” OS kao emulirana 16-bitna aplikacija

- Klijentska aplikacija je sa bazom podataka (DBF tabele¹⁸) komunicirala “File server” režimu, što je činilo komunikaciju u višekorisničkom mrežnom radu neopuzdanom i sporom
- Fizičko ograničenje veličine DBF tabele je bilo 1GB

Korisnički interfejs FMK bio je karakterni - terminal. Iako je to veliki nedostatak sa stanovišta modernog korisnika, taj interfejs je ostavljen i u F18 iz sljedećih razloga:

- “F18” je koncipiran kao nasljednik “FMK” koji će postojeći korisnici moći nastaviti bez dodatne obuke
- Promjena korisničkog interfejsa ne bi bilo moguća bez velikih promjena, što je tražilo dodatne programerske resurse i vrijeme razvoja. Ništa od toga nije bilo dostupno, tako da je postojeći korisnički interfejs od početka uzet kao karakteristika F18, a ne “bug”.
- “F18” je u vrijeme početnog razvoja tretiran kao privremeni projekat koji će biti zamjenjen “xTuple-Postbooks open-source” softverom¹⁹ lokaliziranim za bosansko-hercegovačko tržište.

“F18” je klasična dvoslojna aplikacija za manipulaciju SQL bazom podataka u kojoj se većina logike nalazi na strani klijenta.

7.1.2 “Ad-hoc” rješenja, dugoročne glavobolje

Stabilnost aplikacije u sloju pristupa bazi podataka je i nakon dvije godine razvoja bila upitna. Pokazalo se da je taj sloj aplikacije, napisan na brzinu, loše koncipiran. Funkcija ovog sloja, nazvanog “semafori”²⁰, je bila obezbjediti da aplikativni kôd profunkcioniše nepromjenjen na način da se i dalje lokalno pristupa DBF tabelama koje se u pozadini sinhroniziraju sa SQL bazom. U preliminarnim testovima “semafori” su funkcionalisali dobro. Međutim, stavljanjem u produkciju pokazalo se da je ovaj koncept promašaj u slučajevima većih baza i višekorisničkog pristupa.

7.2 Preduzeće “bring.out”

Preduzeće “bring.out” sa sjedištem u Sarajevu je osnovano u Zenici pod imenom “Sigma-com” 1996 godine. Od početka, bazni proizvod je bio set knjigovodstvenih aplikacija. “bring.out” danas nudi Linux serverska rješenja i knjigovodstvenu aplikaciju F18. Sva programska rješenja “bring.out” su “open-source” softver (nadalje OSS)²¹. “F18” je OSS softver objavljen pod CPAL²² licencom.

7.2.1 Razvojni period, developeri

U periodu 1994-2018 u razvoju su učestvovala četiri developera. Autor ovog rada je osnivač i prvi developer “FMK”, kao i “F18” projekta. Većinu vremena autor ipak nije bio uključen u programerske aktivnosti, nego je obavljao druge poslove.

18 <https://en.wikipedia.org/wiki/DBF>

19 <https://sourceforge.net/projects/postbooks> ; <https://github.com/knowhow/xtuple>

20 https://github.com/knowhow/F18_knowhow/blob/1.7/common/semaphores.prg

21 https://en.wikipedia.org/wiki/Open-source_software

22 <https://opensource.org/licenses/CPAL-1.0>

7.2.2 Kriza projekta

Krajem 2014 godina projekat se našao pred velikim izazovom. Kolega koji je više od deset posljednjih godina bio glavni developer je prestao raditi u preduzeću. Nakon dugogodišnje pauze, autor je preuzeo poslove razvoja. Rezultati u tom periodu nisu bili ohrabrujući. Izvorni kôd je bio nečitljiv, neodržavan, dupliran, nedoslijedan, fragilan. Globalne i privatne varijable koje su izazivale “Variable not found” greške nakon svake promjene na kôd-u. Svaka promjena na projektu je bila frustracija za korisnike. Prijave grešaka: “Ovo je radilo, sada više ne radi?!” bile su svakodnevna pojava. Godine stihiskog i nesistematskog rada na projektu su uzele svoj danak. U preduzeću se počela voditi diskusija da li je rad na projektu uopšte održiv.

7.3 Korištene tehnologije

Iskorištenje postojećeg aplikativnog kôda, multiplatformski pristup te razvoj i korištenje OSS softvera uticale su na odabir tehnologija za razvoj F18.

7.3.1 Harbour programski jezik

Harbour²³ je OSS programski jezik koji je kompatibilan sa “Clipper” programskim jezikom u kome je napisan “FMK”. Harbour pripada familiji dinamičkih jezika kao što su “python” i “ruby”. Nije stekao popularnost van kruga bivših “Clipper” programera. Jezik podržava proceduralni i objektno-orientisani model programiranja. S obzirom da je “FMK” u većini pisan proceduralno, unutar F18 projekta se zadržao proceduralni model. “Harbour” pripada familiji xBase²⁴ jezika. Xbase jezici sadrže integriran jezik za manipulaciju tabelama.

Primjer kreiranja dbf tabele “plate.dbf”:

```
LOCAL nI, aStruct := { ;  
    { "ID",          "N",   8, 0 }, ;  
    { "IME",          "C",  50, 0 }, ;  
    { "PREZIME",      "C",  50, 0 }, ;  
    { "GODINA_RADA", "N",   6, 0 }, ;  
    { "NETO_PLATA",   "N",   8, 2 }, ;  
    { "ZAPOSLEN",     "D",   8, 0 }, ;  
    { "OZENJEN",       "L",   1, 0 }, ;  
    { "MEMO1",         "M",  10, 0 }, ;  
    { "MEMO2",         "M",  10, 0 } }  
  
REQUEST DBFCDX  
  
dbCreate( "plate.dbf", aStruct, "DBFCDX", .T., "PLATE" )
```

Primjer otvaranja i sekvensijalne pretraga “plate.dbf”:

```
? "LOCATE FOR PLATE->OZENJEN .AND. PLATE->NETO_PLATA > 1000"  
WAIT  
dbUseArea( , , "plate.dbf", "PLATE" )  
LOCATE for PLATE->OZENJEN .AND. PLATE->NETO_PLATA > 1000  
DO WHILE PLATE->( Found() )  
    ? PLATE->PREZIME, PLATE->IME, TESTDBF->OZENJEN, PLATE->NETO_PLATA  
    // primjer outputa: HUSREMOVIC ERNAD .T. 1100  
    CONTINUE  
ENDDO
```

23 [https://en.wikipedia.org/wiki/Harbour_\(software\)](https://en.wikipedia.org/wiki/Harbour_(software))

24 <https://en.wikipedia.org/wiki/XBase>

7.3.2 Pristup PostgreSQL bazi iz “harbour”-a

DBF tabele i xBase jezik za manipulaciju podacima nema previše sličnosti sa SQL-om što otežava rad sa relacijskim bazama podataka unutar “harbour”-a. Međutim, “harbour” podržava izradu različitih dodataka za pristup podacima (eng. replacable database drivers - RDD) koji obezbeđuju da se xBase model rada sa podacima primjeni na različite izvore podataka. Za potrebe pristupa PostgreSQL-u bazama se koristi PgSQL RDD²⁵. Bazne funkcije su realizovane unutar F18 “core_sql” komponenti²⁶. Te funkcije omogućavaju da se bazi podataka pristupa idiomatski harbour/xBase programskom jeziku.

Primjer: U niz izvještaja (kartica, lager lista faktura) za pristup fakturama iz određenog perioda se koristi sljedeća sekvenca kôda:

```
find_fakt_za_period( cIdFirma, dDatOd, dDatDo, NIL, NIL, "3" )

? "Fakture za period:", dDatOd, dDatDo
DO WHILE !EOF()
    cIdFirma := fakt->idfirma
    cBrDok := fakt->brdok
    cIdTipDok := fakt->idTipDok

    ? "Fakturna:" cIdFirma, cIdTipDok, cBrDok, " ima sljedeće stavke"
    DO WHILE !EOF .AND. fakt->idfirma == cIdFirma .AND. fakt->idtipdok == cIdTipok ;
        .AND. fakt->brdok == cBrDok
        ? "datum", fakt->datdok, "iznos:", fakt->cijena * fakt->kolicina
    ENDDO

ENDDO
```

Funkcija *find_fakt_za_period()*²⁷ je realizovana uz pomoć core_sql funkcije *use_sql()*²⁸:

```
FUNCTION find_fakt_za_period( cIdFirma, dDatOd, dDatDo, cOrderBy, cWhere, cTag ) 

LOCAL hParams := hb_Hash()

hb_default( @cOrderBy, "idFirma,idtipdok,brdok,rbr" )

IF cIdFirma <> NIL
    IF !Empty( cIdFirma )
        hParams[ "idfirma" ] := cIdFirma
    ENDIF
ENDIF

IF dDatOd != NIL
    hParams[ "dat_od" ] := dDatOd
ENDIF

IF dDatDo != NIL
    hParams[ "dat_do" ] := dDatDo
```

25 <https://github.com/hernad/harbour-core/tree/my-master/contrib/hbpgsql> ;
<https://github.com/hernad/harbour-core/tree/my-master/contrib/sddpg> ; PostgreSQL baza podataka je odabrana iz razloga što je ista baza korištena u “xTuple-Postbooks” projektu.

26 https://github.com/knowhow/F18_knowhow/tree/3-std/core_sql

27 https://github.com/knowhow/F18_knowhow/blob/3-std/fakt/_fakt_sql.prg

28 https://github.com/knowhow/F18_knowhow/blob/3-std/core_sql/use_sql.prg#L159

```

ENDIF

hParams[ "order_by" ] := cOrderBy

hParams[ "indeks" ] := .T.

IF cTag == NIL
  cTag := "1"
ENDIF
hParams[ "tag" ] := cTag

IF cWhere != NIL
  hParams[ "where" ] := cWhere
ENDIF

IF !use_sql_fakt( hParams )
  RETURN .F.
ENDIF

GO TOP

RETURN ! Eof()

FUNCTION use_sql_fakt( hParams )

LOCAL cTable := "fakt_fakt", cAlias := "FAKT"
LOCAL cWhere, cOrder
LOCAL cSql
LOCAL hIndexes, cKey, cTag := "1"

default_if_nil( @hParams, hb_Hash() )

cSql := "SELECT * from fmk." + cTable

cWhere := use_sql_fakt_where( hParams )
cOrder := use_sql_fakt_order( hParams )

IF !Empty( cWhere )
  cSql += " WHERE " + cWhere
  IF !Empty( cOrder )
    cSql += cOrder
  ENDIF
ELSE
  cSql += " OFFSET 0 LIMIT 1"
ENDIF

IF hb_HHasKey( hParams, "alias" )
  cTable := hParams[ "alias" ]
ENDIF

IF hb_HHasKey( hParams, "tag" )
  cTag := hParams[ "tag" ]
ENDIF

SELECT ( F_FAKT )
IF !use_sql( cTable, cSql, cAlias )
  RETURN .F.
ENDIF

IF hParams[ "indeks" ]
  hIndexes := h_fakt_fakt_indexes()

  FOR EACH cKey IN hIndexes:Keys
    INDEX ON &( hIndexes[ cKey ] ) TAG ( cKey ) TO ( cAlias )
  NEXT

```

```

        SET ORDER TO TAG ( cTag )
ENDIF
GO TOP

RETURN .T.

STATIC FUNCTION use_sql_fakt_order( hParams )
LOCAL cOrder := ""

IF hb_HHasKey( hParams, "order_by" )
    cOrder += " ORDER BY " + hParams[ "order_by" ]
ELSE
    cOrder += " ORDER BY idfirma,idtipdok,brdok"
ENDIF

RETURN cOrder

STATIC FUNCTION use_sql_fakt_where( hParams )

LOCAL cWhere := ""
LOCAL dDatOd

IF hb_HHasKey( hParams, "idfirma" )
    cWhere := parsiraj_sql( "idfirma", hParams[ "idfirma" ] )
ENDIF

IF hb_HHasKey( hParams, "idtipdok" )
    cWhere += iif( Empty( cWhere ), "", " AND " ) + parsiraj_sql( "idtipdok", hParams[ "idvn" ] )
)
ENDIF

IF hb_HHasKey( hParams, "brdok" )
    cWhere += iif( Empty( cWhere ), "", " AND " ) + parsiraj_sql( "brdok", hParams[ "brdok" ] )
ENDIF

IF hb_HHasKey( hParams, "idpartner" )
    cWhere += iif( Empty( cWhere ), "", " AND " ) + ;
        parsiraj_sql( "idpartner", hParams[ "idpartner" ] )
ENDIF

IF hb_HHasKey( hParams, "dat_do" )
    IF !hb_HHasKey( hParams, "dat_od" )
        dDatOd := CTOD( "" )
    ELSE
        dDatOd := hParams[ "dat_od" ]
    ENDIF
    cWhere += iif( Empty( cWhere ), "", " AND " ) + ;
        parsiraj_sql_date_interval( "datdok", dDatOd, hParams[ "dat_do" ] )
ENDIF

IF hb_HHasKey( hParams, "where" )
    cWhere += iif( Empty( cWhere ), "", " AND " ) + hParams[ "where" ]
ENDIF

RETURN cWhere

```

Pregledom programskog kôda uočavamo da se unutar funkcije formiraju standardni SQL elementi(ORDER, WHERE), koji se prosljeđuju *use_sql()* funkciji koja na kraju podatke dohvata standardnim SQL upitom. Na ovaj način aplikativni kôd je pretrpio minimalne promjene zamjenom izvora podataka sa DBF tabela (FMK) na relacijsku bazu (F18) .

7.3.3 Više-platformski pristup

“Harbour” ima podršku za sve glavne desktop operativne sisteme. “F18” klijentska aplikacija podržava Linux i Windows klijente.

7.4 Upravljanje projektom

Prethodne sekcije objašnjavaju razloge i ciljeve razvoja “F18”. One ukazuju na poteškoće u održavanju i razvoju projekta. Programski jezik “harbour”, izvorni kôd u lošem stanju, arhitektura stara više od dvadeset godina, te arhaični korisnički interfejs činili su razvoj F18 neatraktivnim.

7.4.1 Poslovni kontekst projekta

Do 2014. godine svi korisnici “FMK” sa manjim bazama podataka migrirali su na “F18”. Time su glavni poslovni ciljevi “F18” ostvareni. Međutim, nestabilnost u sloju baze podataka (ref. Poglavlje: “*Ad-hoc* rješenja, dugoročne glavobolje”) onemogućila je da se proces migracije sa “FMK” na “F18” završi do kraja. Odlazak glavnog developera krajem 2014. godine, činjenica da je projekat općenito neutraktivan za razvoj, doveli su preduzeće u dilemu: “Da li je rentabilno ulagati u ovaj projekat? Da li je bolje da se usmjerimo isključivo na sistemska rješenjima u kojima smo konkurenčni?”

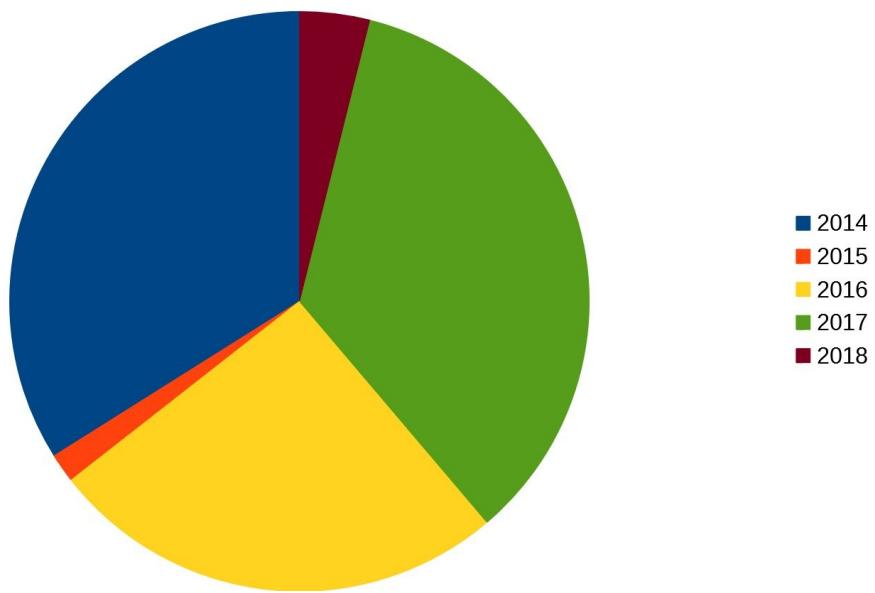
7.4.2 Upotrebnost vrijednost

Zaključeno je da je većina klijenata direktno ili indirektno povezana sa preduzećem preko knjigovodstvenih aplikacija, te da postoji velika opasnost da će konkurenca nakon zamjene knjigovodstvenog softvera ugroziti pozicije preduzeća kod korisnika i po pitanju drugih rješenja i usluga. Dodatno, korisnička baza knjigovodstvenih aplikacija, u kojoj su većina klijenti preko 10 godina, govore u prilog upotreboj vrijednosti projekta. Softver koji funkcioniše decenijama ima bez dvojbe veliku upotrebnu vrijednost. Stoga je odlučeno da se projekat razvoja “F18” nastavi bez obzira na velike nedostatke i probleme.

7.5 Git istorija projekta 2014-2018

Statistika ažuriranja u glavni git repozitorij projekta dobar je pokazatelj dešavanja unutar projekta. Od ukupno preko 5000 “commit”-a, 35% otpada na 2014 godinu unutar koje se dešavao transfer projekta sa starog developera na novog, 2015 došlo je do potpune stagnacije razvoja, dok su se najbitnije aktivnosti desile u periodu 2016-2017 (>60%). U tom periodu je izvršena migracija knjigovodstva posljednjeg, ujedno i najvećeg korisnika.

```
$ git rev-list --count --since="Jan 1 2014" --before="Sep 1 2018" --all => 5024  
$ git rev-list --count --since="Jan 1 2015" --before="Dec 31 2015" --all => 82  
$ git rev-list --count --since="Jan 1 2016" --before="Dec 31 2016" --all => 1288  
$ git rev-list --count --since="Jan 1 2017" --before="Dec 31 2017" --all => 1753  
$ git rev-list --count --since="Jan 1 2018" --before="Sep 30 2018" --all => 200
```



Statistika publikovanja u F18 git repozitorij 2014-2018 god.

Istorija promjena izvornog kôda projekta u periodu na najbolji način oslikava stanje projekta:

- 2014 - odlazeći developer, veliki nivo aktivnosti
- 2015 - projekat neaktivan; dileme oko nastavka projekta
- 2016–2017 – intenzivan razvoj, migracija najvećeg klijenta na F18
- 2018 - period stabilizacije

7.6 Glavni ciljevi razvoja “F18” u periodu 2016-2018

Nakon odluke da se razvoj nastavi, započela je nova etapa razvoja. Otklanjanje ranije dijagnosticiranih problema postavljeno je za primarne ciljeve:

7.6.1 Čitljivost izvornog kôda (eng. source code readability)

- izbrisati komentare koji su zastarjeli ili ne sadrže nikakve korisne informacije
- preimenovati imena funkcija i varijabli da njihova imena ukazuju na namjere programskog kôda
 - npr. `FUNCTION ost_rz()` -> `FUNCTION fin_rucno_zatvaranje_otvorenih_stavki()`
- Eliminisati dijelove koji se više ne koriste
 - funkcije rađene specifično za klijente koji više nisu aktivnisti
 - zakonska rješenja koja više nisu aktuelna (npr. Obračun poreza na promet – zakonska regulativa prije stupanja zakona o PDV-u)

7.6.2 Promjene na sloju pristupa podataka (eng. data layer)

- Komponentu “semafori” (synchronizacija DBF<->SQL) u potpunosti zamijeniti sa novim rješenjem
- Performanse i jednostavnost programerskog interfejsa su prioritet
- Gornji zahtjevi neće omogućiti potpunu kompatibilnost sa FMK aplikativnim kôdom (sa “semaforima” je napravljena ta greška), ali je svakako bitno obezbjediti da potrebne prilagodbe aplikativnog kôda budu minimalne
- U roku od 6 mjeseci na novi način pristupa podacima prebaciti programske module (FIN, KALK, FAKT)

7.6.3 Realizacija programskih zahtjeva korisnika

- U toku 2015 godine došlo je do prekida razvoja. Realizovati zahtjeve koji su ostali aktuelni.
- Krajem 2015 godine ugovorena je migracija posljednjeg korisnika sa “FMK” na “F18”. Proces migracije je sa sobom sadržavao niz zahtjeva na programsku nadogradnju

7.7 Pregled F18, korisnički nivo

Osnovni meni aplikacije – odabir programskog modula:

```
[hernad][f18test_2018]

Tekuća baza: f18test_2018 / db ver: 0.0.28 / nivo log: 9
Korisnik: hernad gr: [] VER: 3.1.214
-----
1. FIN # finansijsko poslovanje
2. KALK # robno-materijalno poslovanje
3. FAKT # fakturisanje
4. ePDV # elektronska evidencija PDV-a
5. LD # obračun plata
6. OS/SII# osnovna sredstva i sitan inventar
7. VIRM # virmanni
8. POS # maloprodajna kasa
-----
S. promjena sezone
B. backup podataka
U. upgrade (nadogradnja) aplikacije
P. parametri aplikacije
W. pregled log-a
X. diag info
```

7.8 Postavke - parametri F18

Jedna od niza formi za podešenje parametara rada aplikacije:

```
Odabir modula za glavni meni ***
FIN: D KALK: D FAKT: D ePDV: D LD: D VIRM: D OS/SII: D POS: D MAT: N RNAL: N

Matični podaci korisnika ***
Puno ime i prezime: [REDACTED]

Email parametri ***
email server: [REDACTED] port: 25
username: [REDACTED]
moja email adresa: [REDACTED]
slati poštu na adresu: [REDACTED]
cc adrese: [REDACTED]

Parametri log-a ***
Briši stavke log tabele starije od broja dana (def. 30): 30

Backup parametri ***
Automatski backup podataka organizacije (interval dana 0 - ne radi ništa): 2
Automatski backup podataka servera (interval 0 - ne radi ništa): 0
Udaljena backup lokacija: [REDACTED]
Ping time kod backup komande: 0

Ostali parametri ***
Dužina stranice za izvještaje (def: 60): 60
BUG report na email (D/N/A/O): A Nivo logiranja (0..9) 9

Kompatibilnost ***
KALK PR: N PTXT: D F18 LO (D/N/O): D F18 updates (D/N): D F18 verzija: 3 - std / S
```

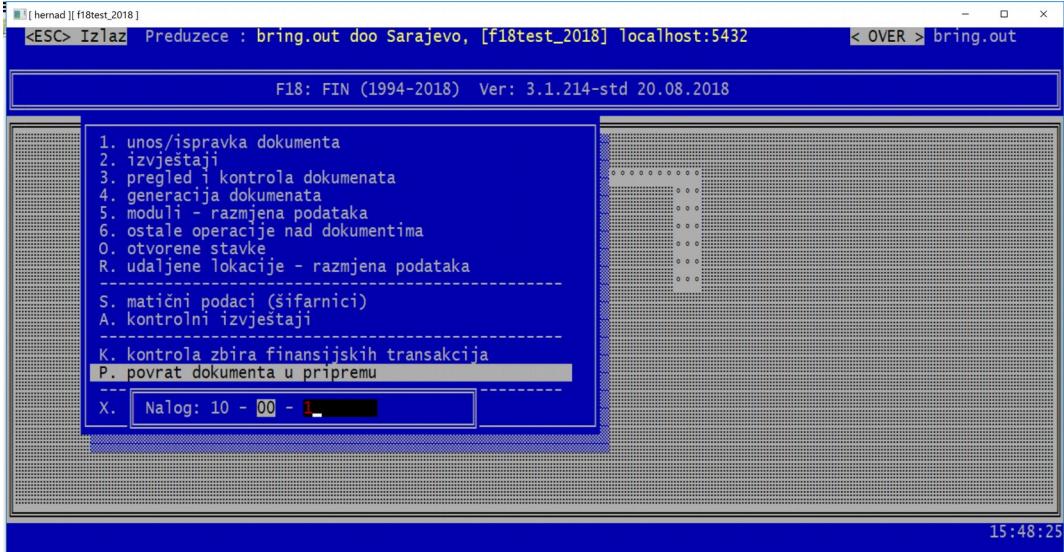
Pored prikazane forme za podešenje zajedničkih parametara, svaki od programskih modula ima svoj set parametara. Sveukupno, aplikacija sadrži 600-700 različitih parametara.

7.9 FIN - Finansijsko poslovanje

Osnovni programski modul je modul za praćenje finansijskog poslovanja. Obezbeđuje unos i ažuriranje finansijskih dokumenata, te standardne finansijske izvještaje (kartice analitičkih konta, kupaca, dobavljača, različite specifikacije, bruto bilans)

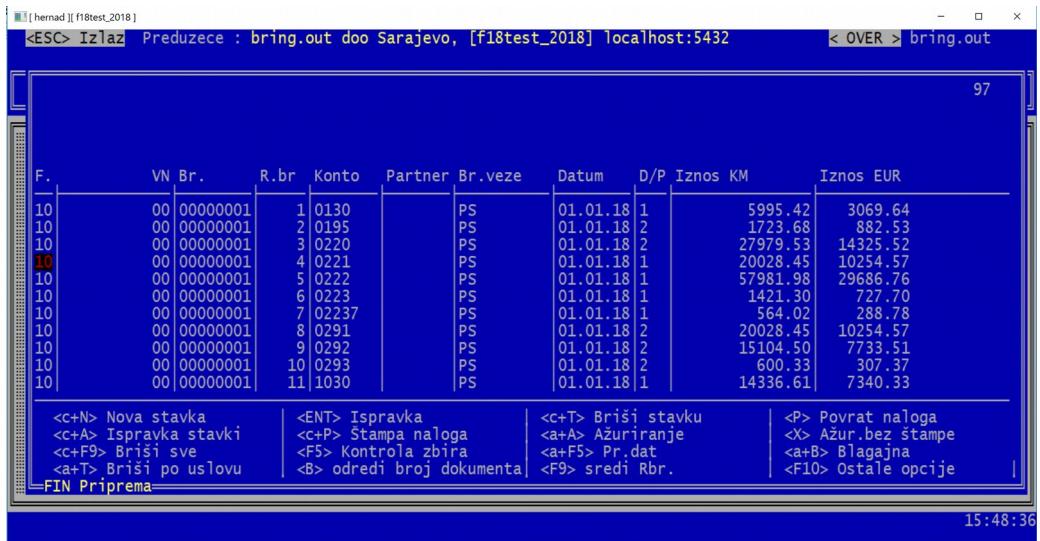
*	R.br	*FIR*	V	BR	* DAT	*	DUGUJE	*	POTRAŽUJE	*	OP.	*
*	MA	N	NAL	NAL	*	NAL	KM	*	KM	*	*	*
1	10	00	00000001	28.02.18			140688.72		140688.72			
2	10	14	00000001	09.02.18			1390.98		1390.98			
3	10	14	00000002	09.02.18			154.71		154.71			
4	10	14	00000003	28.02.18			1248.00		1248.00			
5	10	B2	00000001	28.02.18			692.31		692.31			
6	10	B3	00000001	28.02.18			776.07		776.07			
7	10	I7	00000001	28.02.18			83.75		83.75			
8	10	IB	00000001	28.02.18			29532.42		29532.42			
9	10	IB	00000002	28.02.18			12697.24		12697.24			
10	10	IB	00000003	23.02.18			1157.53		1157.53			
11	10	IB	00000004	28.02.18			5667.70		5667.70			
12	10	IF	00000001	08.02.18			4586.86		4586.86			
13	10	IF	00000002	28.02.18			16595.28		16595.28			
14	10	IF	00000003	28.02.18			2908.16		2908.16			
15	10	OK	00000001	28.02.18			0.03		0.03			

Pregled ažuriranih finansijskih dokumenata (nalog)



FIN: Povrat dokumenta u pripremu

Nalog nakon povrata dokument više ne ulazi u izvještaje. Premješta se u tabelu pripreme, gdje se može korigovati ili bezpovratno izbrisati.

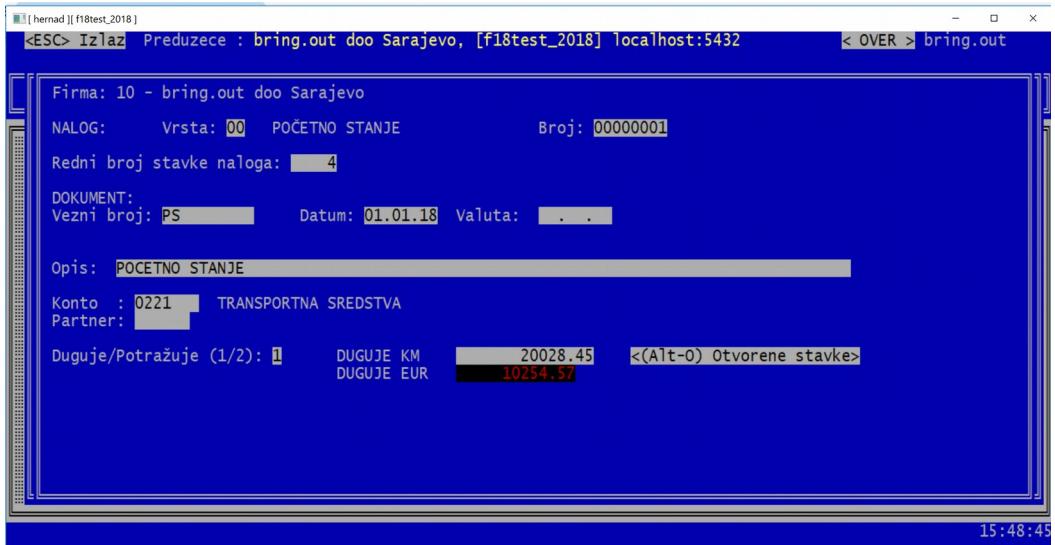


The screenshot shows a terminal window titled 'Izlaz' with the command 'Preduzece : bring.out doo Sarajevo, [f18test_2018] localhost:5432'. The window displays a table of prepared documents with the following columns: F., VN Br., R.br, Konto, Partner, Br.veze, Datum, D/P, Iznos KM, and Iznos EUR. The table contains 12 rows of data. At the bottom of the table, there is a menu with the following options: <c+N> Nova stavka, <ENT> Ispravka, <c+T> Briši stavku, <P> Povrat naloga; <c+A> Ispравка stavki, <c+P> Štampa naloga, <a+A> Ažuriranje, <x> Ažur.bez štampe; <c+F9> Briši sve, <F5> Kontrola zbirke, <a+F5> Pr.dat, <a+B> Blagajna; <a+T> Briši po uslovu, odredi broj dokumenta, <F9> sredi Rbr., <F10> Ostale opcije. The timestamp '15:48:36' is visible at the bottom right of the window.

F.	VN Br.	R.br	Konto	Partner	Br.veze	Datum	D/P	Iznos KM	Iznos EUR
10	00 00000001	1 0130		PS	01.01.18	1		5995.42	3069.64
10	00 00000001	2 0195		PS	01.01.18	2		1723.68	882.53
10	00 00000001	3 0220		PS	01.01.18	2		27979.53	14325.52
10	00 00000001	4 0221		PS	01.01.18	1		20028.45	10254.57
10	00 00000001	5 0222		PS	01.01.18	1		57981.98	29686.76
10	00 00000001	6 0223		PS	01.01.18	1		1421.30	727.70
10	00 00000001	7 02237		PS	01.01.18	1		564.02	288.78
10	00 00000001	8 0291		PS	01.01.18	2		20028.45	10254.57
10	00 00000001	9 0292		PS	01.01.18	2		15104.50	7733.51
10	00 00000001	10 0293		PS	01.01.18	2		600.33	307.37
10	00 00000001	11 1030		PS	01.01.18	1		14336.61	7340.33

<c+N> Nova stavka <ENT> Ispravka <c+T> Briši stavku <P> Povrat naloga
 <c+A> Ispравка stavki <c+P> Štampa naloga <a+A> Ažuriranje <x> Ažur.bez štampe
 <c+F9> Briši sve <F5> Kontrola zbirke <a+F5> Pr.dat <a+B> Blagajna
 <a+T> Briši po uslovu odredi broj dokumenta <F9> sredi Rbr. <F10> Ostale opcije
 =FIN Priprema=

FIN: Tabela pripreme

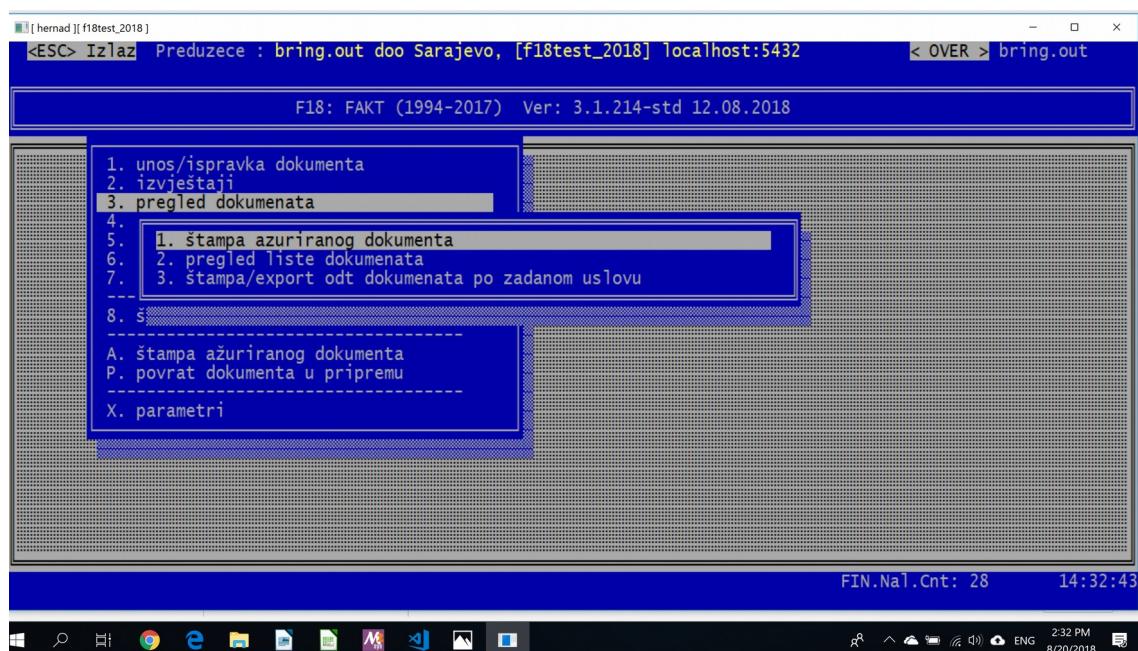


The screenshot shows a terminal window titled 'Izlaz' with the command 'Preduzece : bring.out doo Sarajevo, [f18test_2018] localhost:5432'. The window displays a document detail screen with the following fields: Firma: 10 - bring.out doo Sarajevo, NALOG: Vrsta: 00 POČETNO STANJE, Broj: 00000001, Redni broj stavke naloga: 4, DOKUMENT: Vezni broj: PS, Datum: 01.01.18, Valuta: . ., Opis: POČETNO STANJE, Konto : 0221 TRANSPORTNA SREDSTVA, Partner: , Duguje/Potražuje (1/2): 1, DUGUJE KM: 20028.45, DUGUJE EUR: 10254.57. The timestamp '15:48:45' is visible at the bottom right of the window.

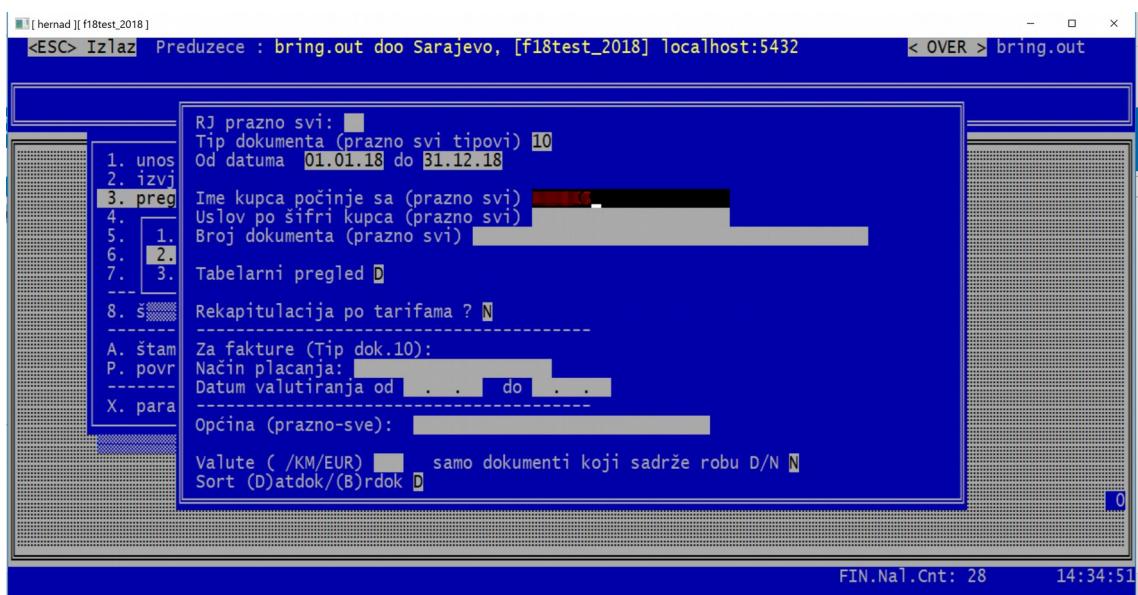
FIN: Stavka naloga u tabeli pripreme

7.10 FAKT - modul za fakturisanje

Komercijalni poslovi se obavljaju kroz ovaj modul. U njemu se evidentiraju ponude i izdaju fakture. Prikaz operacija koje korisniku omogućavaju pregled i štampanje izlaznih faktura:



FAKT: Osnovni meni, podmeni "pregled dokumenata"



FAKT: Forma za unos podataka:
zadavanje željene vrste dokumenata i partnera (tip 10-fakture)

<ESC> Izlaz Preduzece : bring.out doo Sarajevo, [f18test_2018] localhost:5432 < OVER > bring.out

10-10-00030: fiskalni račun: 3602 103

F	RJ	VD	Brdok	VP	Datum	Partner	Ukupno	Rabat
F	10	10	00030		19.02.18		500.000	100.000
F	10	10	00031		19.02.18		140.000	63.000
F	10	10	00033		19.02.18		50.000	0.000

<ENTER> Štampa TXT < P > Povrat dokumenta
 <a+P> ili <L> Štampa ODT < S > Storno dokument
 < R > Štampa fisk.računa < F > ponuda->račun
 < W > Dupliciraj < I > Informacije
 < c+v > Postavi vezu fisk.
 < F5 > osvježi
 < K > Ispravka podataka < T > Duplikat fiskalnog rn.

15:44:20

Tabelarni pregled FAKT dokumenata

out_0001.odt - LO Writer

Datoteka Izmjeni Pogled Ubaci Format Styles Tabela Alati Prozor Pomoc

Tijelo teksta Arial 6

bring.out doo Sarajevo

Adresa: Juraja Najjharta 3, tel: tel: 061477105, 061141311
 ID broj: 4218025900006, PDV broj: 218025900006
 Banke: BBI Banka: 1413065320076611
 Email: office@bring.out.ba, podrška@bring.out.ba

Kupac:
 ..., 72000, ZENICA
 Fax: ..., ID broj: 4 PDV broj: ...
 Broj fiskalnog računa: 3603

POREZNA FAKTURA br. 00031
 Datum dokumenta: 19.02.18
 Datum valute: 19.02.18
 Mjesto: Sarajevo
 Valuta: KM

Narudžba: 29

R.br	Trgovачki naziv dobra/usluge (sifra, naziv)	JMZ	Količina	Cijena bez PDV	Popust (%)	Cijena sa popustom	Ukupno bez PDV
1	...	mj	2.00	70.00	45.00	38.50	140.00

Slovima: devedeset i 9/100 KM

Ukupno bez PDV:	140.00
Ukupno popust:	63.00
Ukupno bez PDV - popust:	77.00
PDV 17%:	13.09
Ukupno sa PDV (KM):	90.09

Fakturisanje po osnovu ugovora o podršci bring.out tješenja.

Page 1 of 1 154 riječi, 1.427 znakova Uobičajeni stil Bosanski Section1 2:37 PM 8/20/2018 100 %

Fakt: pregled fakture u LibreOffice

7.11 Ostali programski moduli

Ostali programski moduli su:

- KALK - Kalkulacije, program za robno/materijalno poslovanje i praćenje proizvodnje
- LD - Obračun plata
- ePDV - evidencija KUF, KIF, obračun PDV-a
- POS - trgovačka kasa na malo sa fiskalnim funkcijama
- OS - evidencija i obračun amortizacije stalnih sredstava i sitnog inventara
- VIRM - unos i štampanje virmana

8 Razvoj projekta “F18” u periodu 2014-2018

8.1 Uvod

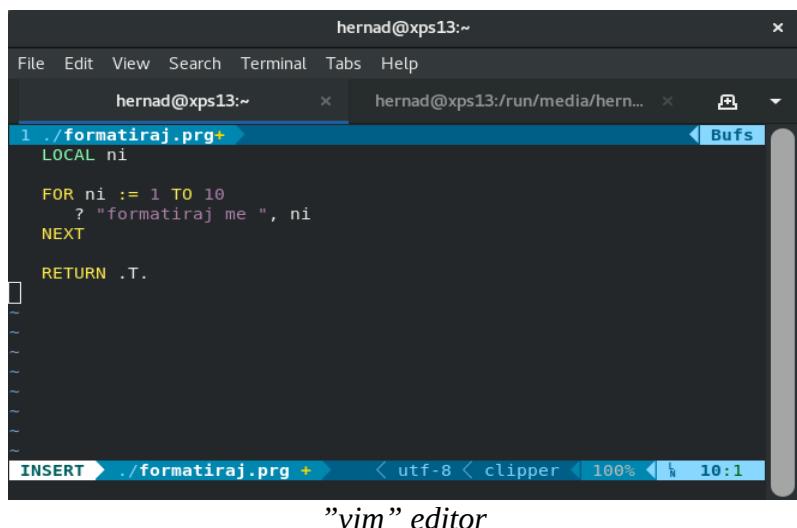
U ovom dijelu rada navedene su glavne aktivnosti i rezultati razvoja projekta “F18” u periodu 2014.-2018. Prikazaćemo najvažnije aktivnosti, te na kraju analizirati realizovano u kontekstu agilnih metoda i praksi.

8.2 “Atom” editor i podrška za “harbour”

Programerski editor je osnovni alat developera. Do 2014. se za razvoj koristio isključivo “vim” editor²⁹ i set unix alata (grep, sed, awk) [SEDAWK]. Najveći problem je bila minimalna podrška editora “harbour” programskom jeziku. S obzirom na nisku popularnost “harbour”-a, moderni programerski editori nisu nudili nikakvu podršku. S druge strane, pojavila se nova generacija programerskih editora, koja je u osnovnoj arhitekturi značajno dodataka za nove programske jezike. “Atom”³⁰ programerski editor identificiran je kao najbolja platforma za razvoj ekstenzija koje će omogućiti veću produktivnosti “harbour” programeru. Rezultat tog rada su tri ekstenzije za atom editor:

- Podrška harbour sintaksi:
 - <https://github.com/hernad/atom-language-harbour>
- Formatiranje harbour koda (hbformat):
 - <https://github.com/hernad/atom-harbour-plus>
- Provjera sintakse i kvaliteta harbour kôda uz pomoć harbour kompjlera:
 - <https://github.com/AtomLinter/linter-harbour>

Ekstenzije su OSS sotver izdane pod MIT³¹ licencom. Napisane su u “coffeescript”³². Autor ovog rada je autor ekstenzija. Dostupne su svakom korisniku “atom” editora kroz opciju “install”:



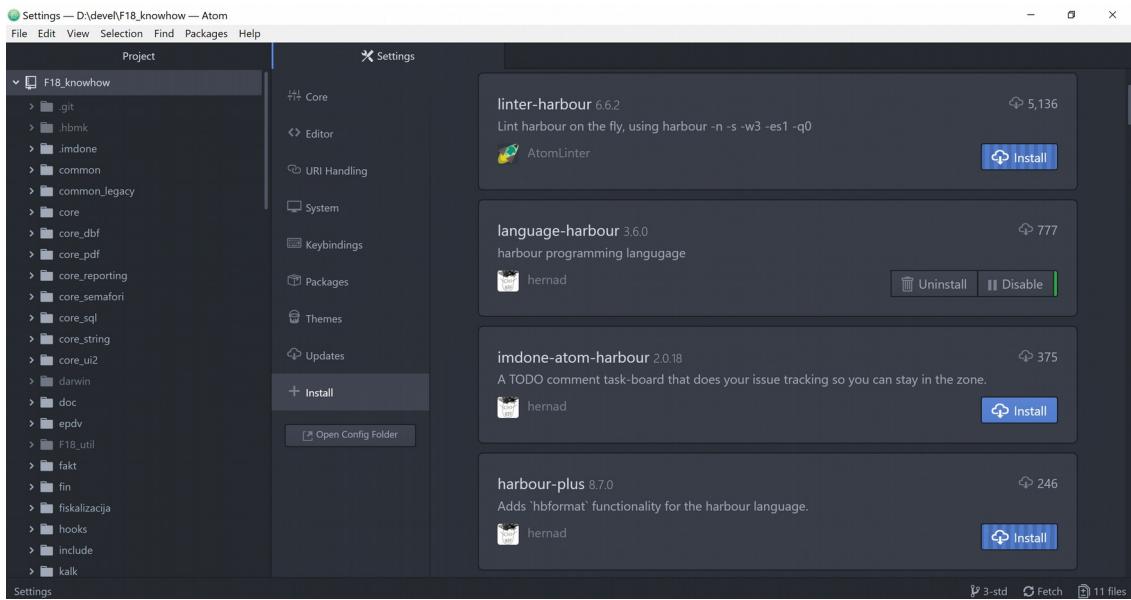
Atom je OSS i multi-platformski softver. Izgrađen je na web tehnologijama.
Njegovo jezgro čini “elektron” <https://electronjs.org>. Bazna komponenta “elektrona” je “google chromium” browser <https://www.chromium.org/Home>

29 <https://www.vim.org>

30 <https://atom.io>

31 <https://opensource.org/licenses/MIT>

32 <https://coffeescript.org/>



"Atom" ekstenzije za harbour (*harbour-linter*, *harbour-language*, *harbour-plus*)

8.2.1 Atom "language-harbour" ekstenzija

Ova ekstenzija obezbjeđuje prepoznavanje sintakse programskog jezika:

The screenshot shows the Atom code editor with the 'core_0.prg' file open in the main pane. The code is written in Harbour language:

```

1  * It is licensed to you under the Common Public Attribution License
2  * version 1.0, the full text of which (including FMK specific Exhibits)
3  * is available in the file LICENSE_CPAL_bring.out_knowhow.md located at the
4  * root directory of this source code archive.
5  * By using this software, you agree to be bound by its terms.
6
7
8
9
10 /*
11
12 #include "f18.ch"
13
14
15 FUNCTION harbour_init()
16
17 rddSetDefault( RDDENGINE )
18 Set( _SET_AUTOPEN, .F. )
19
20 SET CENTURY OFF
21 SET EPOCH TO 1980 // 81 - 1981, 79-2079
22 SET DATE TO GERMAN
23
24 f18_init_threads()
25
26 Set( _SET_OSCODEPAGE, hb_cdpOS() )
27
28 // ? SET( _SET_OSCODEPAGE )
29

```

The status bar at the bottom shows 'core\core_0.prg 1:1', 'LF', 'UTF-8', 'harbour', '3-std', 'Fetch', '11 files', and the system tray with icons for network, battery, and date/time.

Podrška sintaksi "harbour" programskog jezika

8.2.2 Atom “linter-harbour” ekstenzija

Ova ekstenzija je najviše doprinijela produktivnosti developer-a.

“Linter” ekstenzija koji u toku procesa ispravke programskog fajla prikazuje upozorenja (eng. warning, smeđa boja) i greške (eng. error crvena boja). Ekstenzija je time omogućila efikasan refactoring programskog kôda:

```

epdv_sifre_sg.prg — D:\devel\F18_knowhow — Atom
File Edit View Selection Find Packages Help
Project Settings f18_editor.prg clipboard.prg Project Find Results epdv_sifre_sg.prg
75
76 // setuj id tar u kuf/kif
77 • AAdd( aImeKol, { "Set.Pan", {|| s_id_part }, "s_id_part", {|| .T. }, {|| Empty( ws_id_part ) } .OR.
78
79 // setuj id tar u kuf/kif
80 • AAdd( aImeKol, { "Set.Br.Dok", {|| s_br_dok }, "s_br_dok", {|| .T. }, {|| .T. } } )
81
82 • AAdd( aImeKol, { "Aktivan", {|| aktivan }, "aktivan", {|| waktivan := iif( waktivan == " ", "D",
83
84 aKol := {}
85 • FOR i := 1 TO Len( aImeKol )
86 • • AAdd( aKol, i )
87 • NEXT
88
89 RETURN .T.
90
91
16 results found in 11 files for s_br_dok
Finding with Options: Case Insensitive .* Aa
s_br_dok Find All
Replace in project Replace All
File/directory pattern. eg. 'src' to search in the "src" directory or '*.js' to search all JavaScript files
• LF UTF-8 harbour 3-std Fetch 16 f
epdv\epdv_sifre_sg.prg ⌂ 0 ▲ 42 ⌂ 0 85:32

```

“harbour” linter daje sugestije za ispravke koda

Gornji prikaz nam pokazuje primjer djelovanja “lintera”. On daje sugestiju da privatnu varijablu “i” treba deklarisati kao lokalnu. Na dnu ekrana je prikazana rekapitulacija upozorenja i grešaka za programski fajl na kome radimo (42 upozorenja, 0 grešaka). Kada ispravimo kôd prema sugestiji:

```

epdv_sifre_sg.prg — D:\devel\F18_knowhow — Atom
File Edit View Selection Find Packages Help
Project Settings f18_editor.prg clipboard.prg Project Find Results epdv_sifre_sg.prg
40
41
42
43 STATIC FUNCTION set_a_kol( aKol, aImeKol )
44
45 LOCAL i
46
47 aImeKol := {}
48 • AAdd( aImeKol, { "ID", {|| id }, "id", {|| .T. }, {|| .T. } } )
49 • AAdd( aImeKol, { "Opis", {|| naz }, "naz", {|| .T. }, {|| .T. } } )
50
51 // tip: sifrarnik setuje varijable sa "W" prefixom za tekuća polja
52 • AAdd( aImeKol, { "src", {|| src }, "src", {|| .T. }, {|| !Empty( g_src_modul( wSrc, .T. ) ) } } )
53 • AAdd( aImeKol, { "src TD", {|| td_src }, "td_src", {|| .T. }, {|| .T. } } )
54
55
56 // AAdd( aImeKol, { "Src.Lokacija.", {|| s_path }, "s_path", {|| .T. }, {|| .T. } } )
16 results found in 11 files for s_br_dok
Finding with Options: Case Insensitive .* Aa
s_br_dok Find All
Replace in project Replace All
File/directory pattern. eg. 'src' to search in the "src" directory or '*.js' to search all JavaScript files
• LF UTF-8 harbour 3-std Fetch 21 f
epdv\epdv_sifre_sg.prg ⌂ 0 ▲ 31 ⌂ 0 45:11

```

ispravka kôda prema sugestiji “harbour” lintera

The screenshot shows the Atom code editor with a Harbour script file open. The status bar at the bottom indicates the file is 'epdv_sifre_sg.prg'. The code contains several error markers (red dots) indicating ambiguous references. A search bar at the top right shows 's_br_dok' with '16 results found in 11 files'. Below the search bar are 'Find All' and 'Replace All' buttons. The bottom status bar also shows '21 files'.

```

    78 • AAdd( aImeKol, { "Set.Par", {|| s_id_part }, "s_id_part", {|| .T. }, {|| Empty( ws_id_part ) .OR. p_partne
    79
    80 // setuj id tar u kuf/kif
    81 • AAdd( aImeKol, { "Set.Br.Dok", {|| s_br_dok }, "s_br_dok", {|| .T. }, {|| .T. } } )
    82
    83 • AAdd( aImeKol, { "Aktivan", {|| aktivan }, "aktivan", {|| waktivan := iif( waktivan == " ", "D", waktivan
    84
    85 aKol := {}
    86 FOR i := 1 TO Len( aImeKol )
    87     AAdd( aKol, i )
    88
    89 NEXT
    90
    91 RETURN .T.
    92
    93 FUNCTION info_partner()
    94

```

"harbour" linter: Upozorenje "Ambigous reference I" nestaje

Sve ovo se dešava u toku ispravke fajla, tako da developer dobija potrebne informacije trenutno. Dok nije bilo ovog alata svaki refactoring je redovno generisao veliki broj novih programskih grešaka. Zato developeri nisu ni praktikovali ovu tehniku.

8.2.3 Atom editor “harbour-plus” ekstenzija

Unutar ove ekstenzije je realizovano standarizirano formatiranje izvornog kôda uz pomoć harbour “hbformat” programa. Za primjer krećemo od kôda koji je “sirovo” unešen, ne poštujući indentaciju unutar programske petlje, te kapitalizaciju ključnih riječi:

The screenshot shows the Atom code editor with a Harbour script file open. The status bar at the bottom indicates the file is 'formatiraj.prg'. The code has been formatted, showing proper indentation and capitalization. The bottom status bar shows '0 files'.

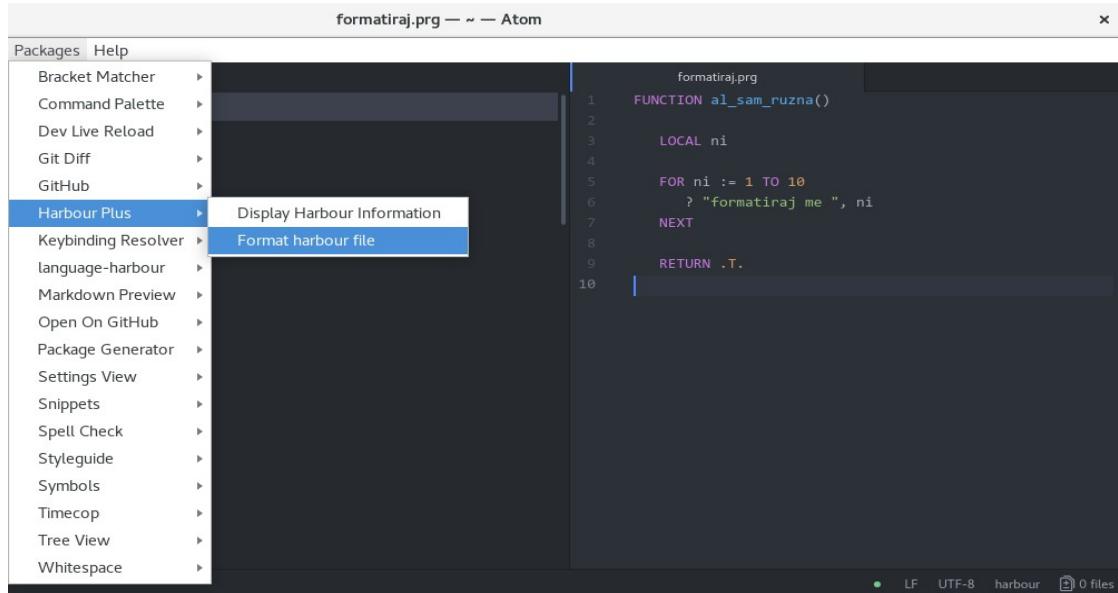
```

1 function al_sam_ruzna()
2 local ni
3 for ni:=1 to 10
4 ? "formatiraj me ", ni
5 next
6 return .t.

```

neformatirani "harbour" programski kôd

Formatiranjem dobijamo potrebnu preglednost - čitljivost izvornog kôda:



```
formatiraj.prg — ~ — Atom
Packages Help
Bracket Matcher
Command Palette
Dev Live Reload
Git Diff
GitHub
Harbour Plus
Keybinding Resolver
language-harbour
Markdown Preview
Open On GitHub
Package Generator
Settings View
Snippets
Spell Check
Styleguide
Symbols
Timecop
Tree View
Whitespace
```

1 FUNCTION al_sam_ruzna()
2
3 LOCAL ni
4
5 FOR ni := 1 TO 10
6 ? "formatiraj me ", ni
7 NEXT
8
9 RETURN .T.
10

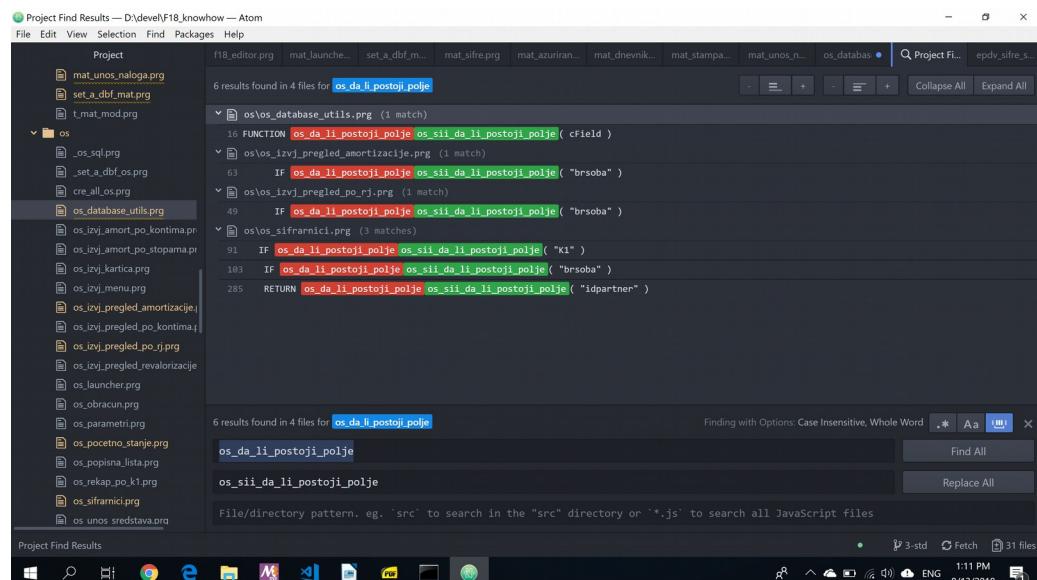
• LF UTF-8 harbour 0 files

"harbour-plus" ekstenzija automatski formatira programski kôd

Ranija praksa, po kojoj je svaki developer formatirao kôd na svoj način, učinila je projekat iznimno "neurednim" (teško čitljivim). Ova opcija pomogla je da se kôd brzo unificira bez puno truda, ali i grešaka koje ručno formatiranje sa sobom nosi.

8.2.4 Atom editor osnovne funkcije

Atom kao moderan editor ima odlično riješene "find-replace" funkcije koje podržavaju "regex" izraze³³. Izmjena naziva funkcije na čitavom projektu je time postao rutinski posao:



"search/replace" funkcije u "atom" editoru

33 https://en.wikipedia.org/wiki/Regular_expression

Nakon promjene treba uočiti da su svi fajlovi koji su pretrpjeli promjene promjenili boju:

The screenshot shows the Atom code editor interface. A file named `os_database_utils.prg` is open, displaying a large amount of PGP source code. The code editor uses color-coded syntax highlighting. A vertical bar on the left side of the code area indicates which files have been modified, with several files under the `os` directory highlighted in yellow. The status bar at the bottom shows various system icons and the date/time.

"atom" editor: identificiranje izvršenih promjena uz pomoć boja

Masovne promjene na projektu su takođe postale jednostavan posao. Promjena sadržaja u 700 fajlova više nije bila dugotrajna i rizična operacija:

The screenshot shows the Atom Project Find Results panel. It displays two search results for the string "Copyright (c) 1994-2011". The results are listed in a tree view, showing the file path and the number of matches. The results are as follows:

- `epdv\epdv_tarifa_utils.prg` (1 match)
- `epdv\epdv_unos_valid.prg` (1 match)
- `epdv\epdv_utils_date.prg` (1 match)
- `epdv\epdv_utils_gen_dok.prg` (1 match)
- `epdv\epdv_utils_kuf_kif.prg` (1 match)
- `epdv\set_a_dbf_epdv.prg` (1 match)
- `epdv\epdv_mod.prg` (1 match)

The status bar at the bottom shows the date/time and the number of files found.

"atom" editor: sugestije prije nego se naprave promjene

Uz ove ekstenzije postiglo se da developer najbitnije operacije obavlja unutar editora, bez potrebe za korištenjem eksternih alata. Složenost korištenja ranijih alata ("vim", "grep", "sed", "awk") je za

većinu developera predstavljala barijeru zbog kojih operacije koje smo demonstrirali ne samo da su bile zahtjevne, nego najčešće nisu ni rađene.

8.2.5 Zaključak

Zahtjevnost razvoja "atom" ekstenzija učinila je da to postane zaseban projekat. Taj podprojekat je uzeo značajno vrijeme i programerske resurse nauštrb direktnog razvoja "F18". Prva izjava u Manifestu, u kojima se navode osnovne vrijednosti agilnog razvoja, glasi: "*Osobe i njihove interakcije više od procesa i alata*".

Da li to znači da smo se previše bavili alatima? Manifest ukazuje na to da su akteri projekta na prvom mjestu. Te osobe su developeri, korisnici, investitori - svi oni koji su na bilo koji način zainteresovani i uključeni u projekat. Atom editor i ekstenzije za "harbour" su obezbjedile produktivnost i ugodniji rad developera. Operacije koje su bile napor sada su nadohvat ruke. Razvoj "F18" je postao puno **ugodniji**. Odluka da se unaprijede razvojni alati ima direktno uporište u sljedećim agilnim praksama i konceptima:

1. stvaranje ugodnog okruženja za developere kroz povećanje produktivnosti
 - developer se više ne žali da je prisiljen da koristi zastarjele alate
2. poticanje razvojnog tima na inovacije i unapređenje u svim elementima djelovanja
 - stvara se svijest unutar razvojnog tima da će se cijeniti svako unapređenje razvojnog procesa, a ne samo direktno izvršenje postavljenih zadataka
3. multi-funkcionalan razvojni tim
 - ako postoji potreba, član tima će se uhvatiti u koštač i sa problemima koji nisu dio njegove ekspertize (u konkretnom slučaju za realizaciju je bilo potrebno ovladati potpuno novim programskim jezikom i okruženjem)

U ovoj analizi ne smiju se izostaviti rizici koje ovakvi projekti sa sobom nose. Nove i nepoznate tehnologije uvjek sa sobom nose dodatni rizik u pogledu ishoda. Kada razvojni tim treba uraditi zadatak koji je sličan nizu prethodnih, pozitivan ishod je izvjestan. Kada se uputi u nepoznatu oblast, rizik neuspjeha je veliki. Da bi posljedice eventualnog neuspjeha bile minimalne, u projektima ove vrste posebnu važnost ima agilna praksa vremensko ograničenje realizacije (eng. **timeboxing**).

8.3 Eliminacija programskog “otpada”

Unapređeni razvojni alati su stvorili preduslove da se izvrši agresivno “čišćenje” programskog kôda projekta. Subjektivni dojam je da je na tom planu učinjeno mnogo. Međutim, rezultati ovih aktivnosti mogu se izraziti i konkretnom metrikom - brojem linija programskog kôda na početku i na kraju ravnog perioda. Željene podatke smo dobili izgenerisali sa “cloc”-om³⁴. Broj programskih fajlova se smanjio neznatno (<10%), ali je broj linija programskog kôda smanjen za skoro 50%! Uklanjanje “mrtvog” i dupliranog koda je dalo značajne rezultate. Uočljivo je “čišćenje” SQL baze: smanjenje sa 73000 na 3000 linija SQL kôda!

8.3.1 Stanje 2014

```
github.com/AlDanial/cloc v 1.72
```

Language	files	blank	comment	code
xBase	821	85575	39690	226491
SQL	2	26820	18864	73769
xBase Header	26	966	320	3759
Bourne Shell	13	137	26	236
DOS Batch	4	78	16	146
Bourne Again Shell	3	63	11	109
SUM:	869	113639	58927	304510

```
$ cat F18_all.cloc.file
```

File	files	blank	comment	code
F18_SQL.cloc	2	26820	18864	73769
F18_kalk.cloc	168	16208	6743	50954
F18_common.cloc	218	20093	11755	47140
F18_ld.cloc	70	8971	3552	26724
F18_fin.cloc	73	9112	3140	26493
F18_fakt.cloc	64	7665	3254	20263
F18_rnal.cloc	67	10024	5040	19607
F18_pos.cloc	66	5215	2644	15100
F18_epdv.cloc	39	2765	1638	5783
F18_mat.cloc	24	1918	591	5651
F18_kadev.cloc	21	1919	636	5537
F18_os.cloc	23	1528	560	4380
F18_virm.cloc	15	1125	457	2620
F18_scripts.cloc	19	276	53	489
SUM:	869	113639	58927	304510

34 <https://github.com/AlDanial/cloc>

8.3.2 Stanje 2018

```
$ cat F18_all.cloc.lang
github.com/AlDanial/cloc v 1.72
```

Language	files	blank	comment	code
xBase	740	70869	23820	171308
SQL	4	1808	1540	2971
xBase Header	18	698	571	2577
Bourne Shell	32	262	67	522
Markdown	4	194	0	459
DOS Batch	6	100	39	171
Bourne Again Shell	3	63	11	109
SUM:	807	73994	26048	178117

```
$ cat F18_all.cloc.file
```

File	files	blank	comment	code
F18_kalk.cloc	141	14476	4686	37531
F18_common.cloc	184	13951	5711	30957
F18_fin.cloc	98	9683	3001	25253
F18_ld.cloc	64	8053	2344	20606
F18_fakt.cloc	61	7147	2305	17117
F18_pos.cloc	60	5249	1893	13241
F18_core.cloc	46	2855	679	6249
F18_epdv.cloc	38	2852	1002	5761
F18_mat.cloc	21	1947	620	5454
F18_os.cloc	21	1706	610	4419
F18_fiskalizacija.cloc	9	2031	1002	3915
F18_SQL.cloc	6	1819	1542	2990
F18_virm.cloc	14	1061	319	2094
F18_core_semafori.cloc	7	780	248	1787
F18_scripts.cloc	37	384	86	743
SUM:	807	73994	26048	178117

8.3.3 Zaključak

Dobijeni rezultati potvrđuju da su se desile radikalne promjene strukture izvornog kôda. Refaktoring i “čišćenje” nepotrebnog koda (eng. cleanup) učinile su projekat značajno jednostavnijim za održavanje i nadogradnju. Ove aktivnosti su u kontekstu agilnog razvoja softvera realizacija principa 8 koji navode tvorci Manifesta: “*Agilni procesi promoviraju održivi razvoj softvera. Investitori, developeri, korisnici trebaju moći nastaviti učešće u održavanju softvera neograničeno.*”

“F18” je zbog poduzetih aktivnosti postao projekat koji se uz manje troškove može održavati i nadograđivati.

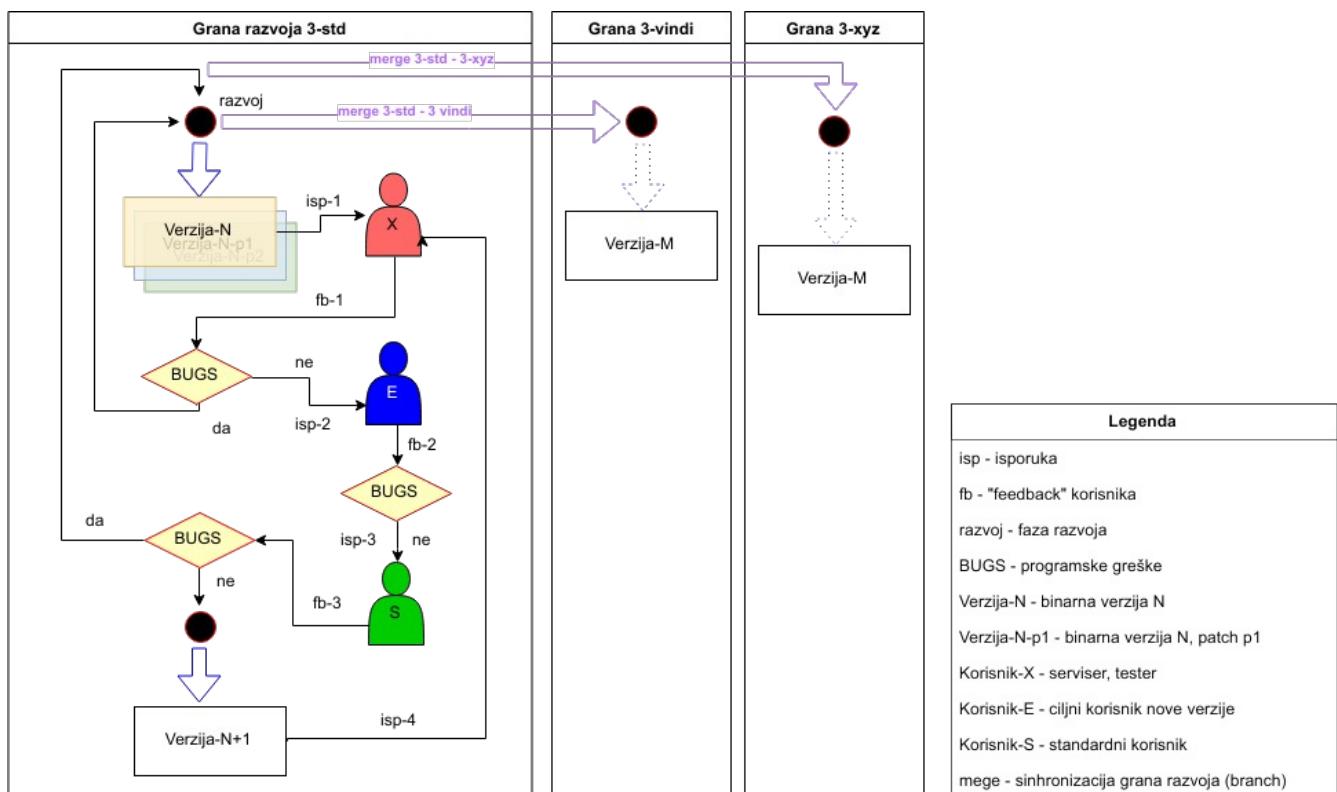
Što se tiče pojedinačnih metoda agilnog razvoja, podsjetimo se da je centralna ideja “*Lean software development*”-a upravo eliminacija “otpada”. Stoga se može reći da su primjenjene “lean” tehnike.

8.4 “F18 update” - provjera i instalacija novih verzija

U posljednjoj liniji zajednički parametara, navedeno je da klijent kontroliše dostupnost novih verzija, te da koristi granu “3-std” (eng. git-branch), standardnu verziju (mogući izbori su S-standardna verzija, E-edge, X-eksperimentalna verzija). Prilikom prvog pokretanja aplikacija vrši se provjera stanja na “github” serveru:



Ovaj koncept automatske instalacije novih verzija se pokazao veoma bitnim. On je omogućio da samo ciljana grupa korisnika dobija najnovije verzije (E), dok većina dobija verzije sa izvjesnim zakašnjenjem (S). Period “Razvoja -> isporuka nove verzije” je u većini slučajeva značajno smanjen i pojednostavljen. Većina korisnika(S) dobija stabilne verzije, s obzirom da se glavni trijaž grešaka dešava unutar male grupe korisnika (X, E).



F18 iteracije - isporuka novih verzija korisnicima, tok operacija (eng. deployment workflow)

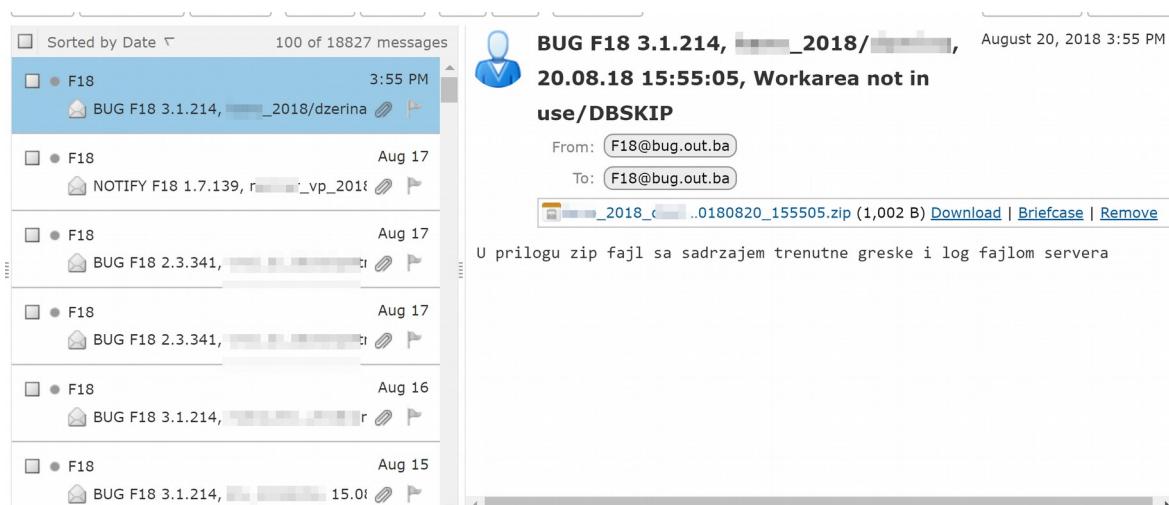
8.4.1 Zaključak

Ova opcija je direktno pomogla agresivnijoj i manje “bolnoj” isporuci novih verzija (sa aspekta korisnika, ali i razvojnog tima). Ona je omogućila da se nove opcije isporučuju svakodnevno, nerijetko više puta na dan. Prije realizacije ove opcije, takva frekvencija izdavanja novih verzija jednostavno nije bila moguća. U poglavlju “Organizacija XP tima” je naglašena **iterativnost razvoja** kao temeljna praksa kojom se postiže agilnost razvoja. Svaka od agilnih metoda ukazuje na temeljnu važnost iterativnog pristupa. To je i očekivano, uvezvi u obzir da je princip 1. tvoraca Manifesta: “*Naš najveći prioritet je zadovoljiti klijenta kroz rane i kontinuirane isporuke funkcionalnog softvera*”.

Pored toga, gornji dijagram toka, ukazuje na intenzivne i kratke “feedback” petlje između razvoja i korisnika.

8.5 F18 automatski izvještaji o greškama (eng. bug reports)

Na početku 2014 je dizajnirana i realizovana opcija automatske prijave grešaka³⁵. Ova opcija na “bring.out” email server šalje izvještaje greškama. U periodu 2014-2018 na server je stiglo skoro **19000 bug reporta**:



Svaki bug report sastoji se od dva priloga:

- Isječak serverskog log-a
- Lista poziva koji su uzrokovali grešku (eng. “call stack”) i stanje bitnih parametara u toku greška

Pogledajmo sadržaje jedne greške pristigle od korisnika:

BUG F18 3.1.214, firmatest_2018/korisnik4, 16.08.18 14:12:38, Zero divisor//

U naslovu greške nalazi se verzija (3.1.214), baza podataka i poslovna godina (firmatest, 2018), ime korisnika kod koga se desila greška (korisnik4), datum i vrijeme, te poruka o grešci.

³⁵ https://github.com/knowhow/F18_knowhow/blob/3-std/common/bug_report.prg

8.5.1 Serverski log

Izgled serverskog log-a:

PREGLED LOG-a

```
-----  
Datum / vrijeme      Korisnik    operacija  
-----  
  
2018-08-16 12:12:34 korisnik4    GLOBALERRORHANDLER(181) : BUG REPORT: Verzija programa: 3.1.214/5.10.0  
                                KALK_STAMPA_DOK_RN / 141 // 3 KALK_STAMPA_DOKUMENTA / 198 ....  
2018-08-16 12:06:09 korisnik4    KALK_AZUR_SQL(713) : F18_DOK_OPER: ažuriranje kalk dokumenta: 10-16-G0087/ZS  
2018-08-16 12:06:07 korisnik4    FIN_AZURIRANJE_NALOGA(110) : F18_DOK_OPER: azuriranje fin naloga: 10-96-00087/ZS  
2018-08-16 12:06:01 korisnik4    KALK_AZUR_SQL(713) : F18_DOK_OPER: ažuriranje kalk dokumenta: 10-96-00087/ZS  
2018-08-16 12:04:21 korisnik4    FIN_AZURIRANJE_NALOGA(110) : F18_DOK_OPER: azuriranje fin naloga: 10-UF-00000188  
2018-08-16 12:04:09 korisnik4    FIN_NALOG_BRISI_IKUMULATIVA(165) : F18_DOK_OPER: POV RAT_FIN: 10-UF-00000188  
...  
2018-08-16 11:46:41 korisnik4    KALK_AZUR_SQL(713) : F18_DOK_OPER: ažuriranje kalk dokumenta: 10-10-00139/ZS  
2018-08-16 11:46:20 korisnik4    KALK_AZUR_SQL(713) : F18_DOK_OPER: ažuriranje kalk dokumenta: 10-10-00138/ZS  
2018-08-16 11:39:50 korisnik3    FIN_AZURIRANJE_NALOGA(110) : F18_DOK_OPER: azuriranje fin naloga: 10-66-B0000208  
2018-08-16 11:38:10 korisnik3    FIN_AZURIRANJE_NALOGA(110) : F18_DOK_OPER: azuriranje fin naloga: 10-66-B0000209  
2018-08-16 11:36:05 korisnik3    FIN_AZURIRANJE_NALOGA(110) : F18_DOK_OPER: azuriranje fin naloga: 10-66-B0000210  
...  
2018-08-16 10:49:58 korisnik1    KALK_POVRAT_DOKUMENTA(125) : F18_DOK_OPER: KALK DOK_POV: 10-81-00256/BP  
2018-08-16 10:41:08 korisnik4    FIN_AZURIRANJE_NALOGA(110) : F18_DOK_OPER: azuriranje fin naloga: 10-IF-00000246  
2018-08-16 10:36:55 korisnik4    FIN_AZURIRANJE_NALOGA(110) : F18_DOK_OPER: azuriranje fin naloga: 10-61-ML000152  
2018-08-16 10:23:22 korisnik4    FAKT_BRISANJE_PRIPREME(650) : F18_DOK_OPER: fakt, brisanje dokumenta iz pripreme: 10-20-00009  
2018-08-16 10:22:29 korisnik4    POV RAT_FAKT_DOKUMENTA(153) : F18_DOK_OPER: fakt povrat dokumenta u pripremu: 10-20-00009  
...  
2018-08-16 10:20:20 korisnik4    BROWSE_SNIMI_PROMJENE_SIFARNIKA(979) : F18_DOK_OPER: dodavanje/ispravka zapisa u sifarnik roba  
...  
2018-08-16 10:17:04 korisnik4    BROWSE_SNIMI_PROMJENE_SIFARNIKA(1002) : F18_DOK_OPER: NOVI TBL: roba NEW: ID=5010#MATCH_CODE=#SIFRADOB=#NAZ=Test Artikal XYZ#JM=KG#IDTARIFA=PDV0#NC=0.00#VPC=11.73#MPC=11.- ...  
2018-08-16 10:17:04 korisnik4    BROWSE_SNIMI_PROMJENE_SIFARNIKA(979) : F18_DOK_OPER: dodavanje/ispravka zapisa u sifarnik roba  
2018-08-16 10:07:03 korisnik4    BROWSE_SNIMI_PROMJENE_SIFARNIKA(1002) : F18_DOK_OPER: NOVI TBL: partn NEW: ID=P-  
                                BAND#MATCH_CODE=#NAZ="TEST  PREDUZECE"  D00  SARAJEVO#NAZZ=#PTT=71000#MJESTO=SARAJEVO#ADRESA=SARAJEV-0  STARIGRAD#ZIROR=#REJON=#TELEFON=099/110-3933#DZIROR=#FAX=#MOBTEL=#IDOPS=#_KUP=#_DOB=#_BANKA=#_RA-DNIK=#IDREFER=#  
....
```

```

2018-08-16 06:43:18 korisnik1      EDIT_FIN_PRIPR_KEY_HANDLER(386) : F18_DOK_OPER: fin, brisanje stavke u pripremi:
10-61-BP000160      stavka br:    14

2018-08-16 06:36:20 korisnik1      FIN_AZURIRANJE_NALOGA(110) : F18_DOK_OPER: azuriranje fin naloga: 10-61-T0000159

2018-08-16 06:31:33 korisnik2      KALK_AZUR_SQL(713) : F18_DOK_OPER: ažuriranje kalk dokumenta: 10-42-00221/5

```

8.5.2 “Call stack” i stanje sistema u trenutku greške

Ovaj izvještaj sadrži:

- “call stack”-a
- sadržaj tabele koja je bila otvorena u toku greške (KALK_PRIPR)
- niz podataka o serveru (verzija, relevantne IP adrese, itd.)

```

F18 bug report (v6.0) : 16.08.18 14:12:35
=====
Verzija programa: 3.1.214/5.10.0 27.06.2018

SubSystem/severity : BASE      2
GenCod/SubCode/OsCode :      5     1340      0
Opis             : Zero divisor
ImeFajla        :
Operacija       : /
Argumenti        : _?_
canRetry/canDefault : .f. .f.

----- SERVER connection info: -----
host/database/port/schema : 192.168.59.10 / maksumic_2018 /
5432 / public
user : korisnik4

BUG REPORT: Verzija programa: 3.1.214/5.10.0 27.06.2018 ;
SubSystem/severity : BASE      2 ;
GenCod/SubCode/OsCode :      5     1340      0 ;
Opis: Zero divisor ; ImeFajla           : ;
Operacija: / ; Argumenti          : _?_ ;
canRetry/canDefault : .f. .f. ;
CALL STACK:
1 (b)F18_ERROR_BLOCK / 52
2 KALK_STAMPA_DOK_RN / 141
3 KALK_STAMPA_DOKUMENTA / 198
4 KALK_PRIPR_KEY_HANDLER / 235
5 (b)KALK_PRIPR_OBRADA / 141
6 MY_BROWSE_F18_KOMANDE_WITH_MY_KEY_HANDLER / 654
7 MY_BROWSE / 337
8 KALK_PRIPR_OBRADA / 141
9 (b)TKALKMOD_PROGRAMSKI_MODUL_OSNOVNI_MENU / 57

F18 client required server db >=      : 0.0.25 / 25
Actual knowhow ERP server db version : 0.0.27 / 27
----- BEGIN PostgreSQL vars -----
server_version          : 9.5.1
TimeZone                : UTC
----- END PostgreSQL vars -----
client_addr              : 192.168.59.114
client_port               : 53716
server_addr              : 192.168.59.10
server_port               : 5432
user                     : korisnik4

```

/----- END PostgreSQL sys info -----/

Trenutno radno područje: KALK_PRIPR , record: 7 / 11
Record content:

1	IDFIRMA	C	2	0 10	27	MPCSAPP	N	18	8	0.00000000
2	IDROBA	C	10	0 9960	28	IDTARIFA	C	6	0	PDV17
3	IDKONTO	C	7	0 1200	29	MKONTO	C	7	0	1200
4	IDKONTO2	C	7	0 1100	30	PKONTO	C	7	0	0
5	IDZADUZ	C	6	0	31	MU_I	C	1	0	1
6	IDZADUZZ	C	6	0 087/18	32	PU_I	C	1	0	0
7	IDVD	C	2	0 RN	33	ERROR	C	1	0	0 0
8	BRDOK	C	8	0 ,0087/PP	34	KOLICINA	N	18	8	3.52000000
9	DATDOK	D	8	0 11.08.18	35	GKOLICINA	N	18	8	0.00000000
10	BRFAKTP	C	10	0 087/18	36	GKOLICIN2	N	18	8	0.00000000
11	DATFAKTP	D	8	0 11.08.18	37	FCJ	N	18	8	0.00000000
12	IDPARTNER	C	6	0	38	FCJ2	N	18	8	0.00000000
13	RBR	C	3	0 6	39	FCJ3	N	18	8	0.00000000
14	PODBR	C	2	0	40	RABAT	N	18	8	0.00000000
...					...					
21	TRABAT	C	1	0	46	ZAVTR	N	18	8	0.00000000
22	TMARZA	C	1	0 A	47	MARZA	N	18	8	-9999.00000000
23	TMARZA2	C	1	0	48	MARZA2	N	18	8	0.00000000
24	NC	N	18	8 9999.00000000	49	RABATIV	N	18	8	0.00000000
25	MPC	N	18	8 0.00000000	50	VPCSAP	N	18	8	0.00000000
26	VPC	N	18	8 0.00000000						

18 MAIN / 45
== END OF BUG REPORT ==

8.5.3 Svrha i značaj automatskih “bug report”-a

Motiv za izradu automatskih bug-reporta je bila činjenica da je do tada serviseru bilo potrebno 30-45 minuta da prikupi podatke o grešci. Povećani razvoj redovno rezultira povećanjem programskih grešaka. U najvećem broju slučajeva, greška bi bila otklonjena u kratkom roku nakon što se dobiju podaci o grešci. Međutim, ručno prikupljanje tih podataka bi se često znalo desiti tek nekoliko dana nakon prijave problema. To je stvaralo dva glavna problema:

- serviser je izložen velikom pritisku
- korisnici nezadovoljni sporim rješavanjem problema

Oba problema su eliminisana automatskim bug reportima. Napravimo jednostavnu ekonomsku računicu efekata ove opcije:

Sa pretpostavkom da su pristigli bug report-i imali koeficijent ponavljanja 4, te da prosječno ručno prikupljanje traje 1/2 h, direktna ušteda je:

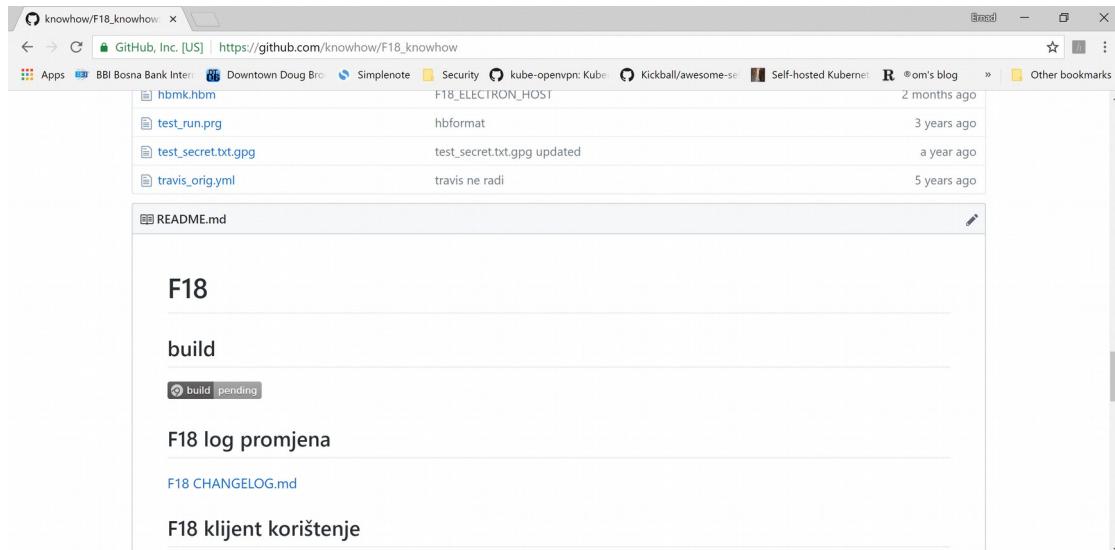
$19000/4*0,5 = 2375$ serviser/h. Kada to podijelimo na 4 godine, svaki mjesec je smanjen angažman servisera za 50 h. Može se zaključiti da su automatski reporti proizveli korist koja odgovara angažmanu dodatnog servisera!

8.5.4 Zaključak

Veliki dio zaključaka koji se odnose na ranije navedenu opciju “F18 update” su identični u slučaju ove opcije. Ove dvije opcije su svakako komplementarne i usko povezane. Dodajmo tome stavku jezikom “Lean software development”-a da je uz pomoć ove opcije eliminisan “otpad” nepotrebnih operacija servisera.

8.6 F18 automatska integracija (CI)

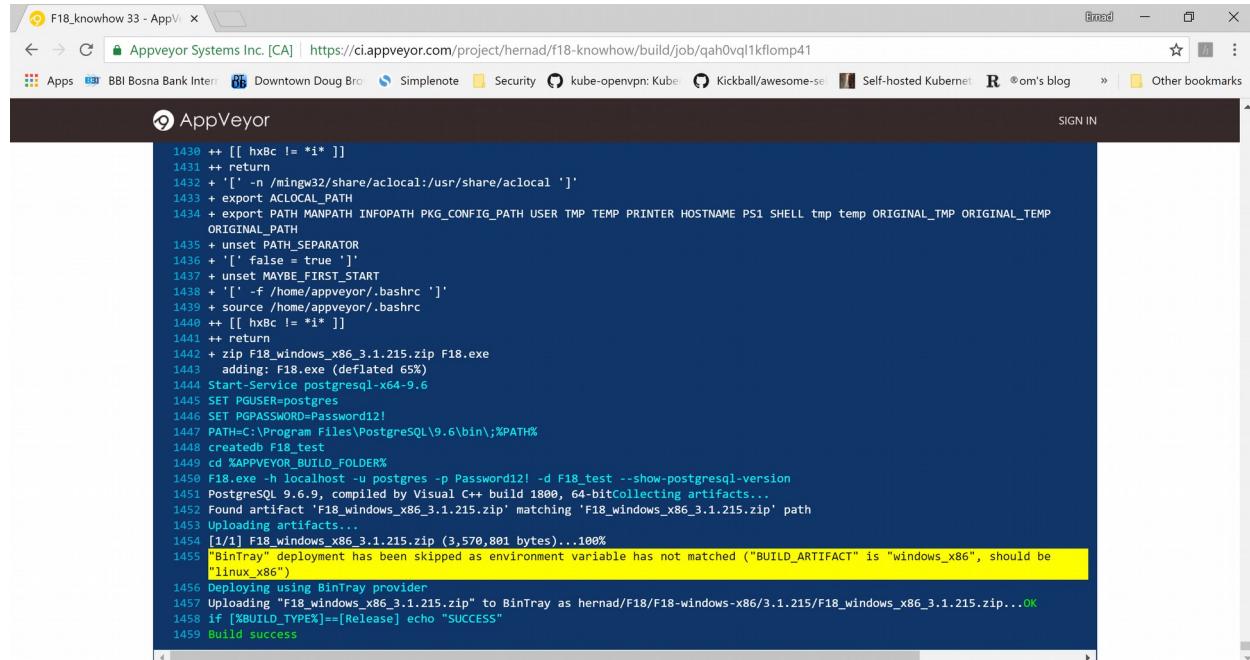
Automatska integracija (eng. Continous integration - CI) koja se koristi u "F18" obezbjeđuje da se nakon publikovanja nove verzije projekta na "github", aktivira automatsko kreiranje Linux i Windows verzije. Kao CI platforma koristi se "Appveyor" servis³⁶:



The screenshot shows a GitHub repository named 'knowhow/F18_knowhow'. The repository has several files listed in the sidebar: 'hbmc.hbm', 'F18_ELECTRON_HOST', 'test_run.prg', 'hbformat', 'test_secret.txt.gpg', 'test_secret.txt.gpg updated', 'travis_orig.yaml', 'travis ne radi', 'README.md', 'F18', 'build', 'F18 log promjena', 'F18 CHANGELOG.md', and 'F18 klijent korištenje'. A 'build pending' status badge is visible next to the 'build' link. The repository was last updated 2 months ago.

Kreiranje (eng. build) nove verzije je započelo, što se vidi po "build pending" statusu

Kada se na git rezervorij na "github"-u publikuje nova verzija (sa git "tag"-om i predefinisanom porukom "BUILD_RELEASE XYZ") započinje build proces a "appveyor" serverima. Ako je "build" proces uspješan, artifakt se publikuje na "bintray":



The screenshot shows an AppVeyor build log for project 'F18_knowhow' (job ID: qah0vql1kflopmp41). The log output is as follows:

```
1430 ++ [[ $hxBC != *i* ]]
1431 ++ return
1432 + '[' -n /mingw32/share/aclocal:/usr/share/aclocal ']'
1433 + export ACLOCAL_PATH
1434 + export PATH MANPATH INFOPATH PKG_CONFIG_PATH USER TMP TEMP PRINTER HOSTNAME PS1 SHELL tmp temp ORIGINAL_TMP ORIGINAL_TEMP ORIGINAL_PATH
1435 + unset PATH_SEPARATOR
1436 + '[' false = true ']'
1437 + unset MAYBE_FIRST_START
1438 + '[' -f /home/appveyor/.bashrc ']'
1439 + source /home/appveyor/.bashrc
1440 ++ [[ $hxBC != *i* ]]
1441 ++ return
1442 + zip F18_windows_x86_3.1.215.zip F18.exe
1443 adding: F18.exe (deflated 65%)
1444 Start-Service postgresql-x64-9.6
1445 SET PGUSER=postgres
1446 SET PGPASSWORD=Password12!
1447 PATH=C:\Program Files\PostgreSQL\9.6\bin;%PATH%
1448 createdb F18_test
1449 cd %APPVEYOR_BUILD_FOLDER%
1450 F18.exe -h localhost -u postgres -p Password12! -d F18_test --show-postgresql-version
1451 PostgreSQL 9.6.9, compiled by Visual C++ build 1800, 64-bit
1452 Found artifact 'F18_windows_x86_3.1.215.zip' matching 'F18_windows_x86_3.1.215.zip' path
1453 Uploading artifacts...
1454 [1/1] F18_windows_x86_3.1.215.zip (3,570,801 bytes)...100%
1455 "BinTray" deployment has been skipped as environment variable has not matched ("BUILD_ARTIFACT" is "windows_x86", should be "linux_x86")
1456 Deploying using BinTray provider
1457 Uploading "F18_windows_x86_3.1.215.zip" to BinTray as hernad/F18/F18-windows-x86/3.1.215/F18_windows_x86_3.1.215.zip...OK
1458 if [%BUILD_TYPE%]==[Release] echo "SUCCESS"
1459 Build success
```

36 <https://www.appveyor.com>; Appveyor je besplatan za OSS projekte.

“Bintray”³⁷ je servis na koji se publikuju binarne verzije:

The screenshot shows a web browser window with the URL <https://bintray.com/hernad/F18>. The page displays a search bar and a sorting dropdown set to "Sorted By Name". Below this, there are four package cards:

- F18-linux-x86**: Version 3.1.214 (Jun 27, 2018) by Ermad Husremović
- F18-windows-x86**: Version 3.1.215 (Aug 13, 2018) by Ermad Husremović
- postgresql-windows-x86-dlls**: Version 3.1.214 (Jun 27, 2018) by Ermad Husremović
- template**: Version 3.1.214 (Jun 27, 2018) by Ermad Husremović

A "Feedback" button is visible on the right side of the page.

web stranica “bintray” servisa, nakon što je build nove windows verzije (3.1.215) završen.

Identičnim “build” procesom se kreiraju i linux verzije. Po uspješnom okončanju čitavog procesa, “appveyor” kontrolna ploča daje status “zeleno”:

The screenshot shows the AppVeyor CI dashboard for the project "F18_knowhow". It features a "LATEST BUILD" card for "BUILD_RELEASE 3.1.215", which was triggered 12 minutes ago by Ermad Husremović. The build status is green, indicating success. The card shows metrics: 33 tests, 3 minutes ago in 8 min 22 sec, and a "JOBS" button. Below the card is a table of build jobs:

JOB NAME	TESTS	DURATION
Environment: APPVEYOR_BUILD_WORKER_IMAGE=Visual Studio 2015, MSYS2_ARCH=i686, MSYSTEM=MINGW32, BUILD_A...	4 min 5 sec	3 minutes ago in 8 min 22 sec
Environment: APPVEYOR_BUILD_WORKER_IMAGE=Ubuntu, BUILD_ARTIFACT=linux_x86, appveyor_repo_tag=true	2 min 7 sec	

"Appveyor" kontrolna ploča na kraju uspješnog "build" procesa

37 Slično “appveyor”-u, “bintray” je besplatan za “open-source” projekte.

8.6.1 Zaključak

Nekoliko minuta nakon publikovanja izvornog kôda nove verzije su dostupne korisnicima. Automatizirani proces kreiranja binarnih verzija obezbjeđuje redovno integrise tehnike automatskog testiranja prije publikovanja novih verzija. Iako "F18" nije pokriven detaljnim funkcionalnim testovima (nisu prakticirane TDD tehnike), appveyor³⁸ skripta provjerava da li kreirana verzija uspješno komunicira sa database serverom:

```
test_script:  
    - cmd: cd %APPVEYOR_BUILD_FOLDER%  
  
Windows => - cmd: F18.exe -h localhost -u postgres -p Password12! -d F18_test --show-postgresql-version  
  
Linux => - sh: LD_LIBRARY_PATH=. xvfb-run --server-args="-screen 0 1024x768x24" ./F18 -h localhost -u postgres -p Password12! -d F18_test --show-postgresql-version
```

Bez obzira na jednostavnost, ovaj test garantuje da se korisniku neće isporučiti verzija koja ne može komunicirati sa bazom podataka, što je od suštinske važnosti. Automatsko kreiranje novih binarnih verzija (CI - eng. continuous integration) je tehnika koja je u svim agilnim metodama visoko preporučena, a u XP obavezna. U "*Lean software development*" metodi CI je praksa koja razvojni tim rješava nepotrebnih operacija manuelne isporuke nove verzije.

U kontekstu projekta "F18", očigledna je povezanost ove opcije sa "F18 update" opcijom. Nije zvoreg ponoviti da bi bez ovih tehnika **iterativni razvoj** bio praktično nemoguć.

38 https://github.com/knowhow/F18_knowhow/blob/3-std/appveyor.yml

9 Iz osobne prakse

9.1 Software koji se ne koristi je otpad

Softverska industrija svakodnevno izbacuje nove alate i tehnologije. Upoznavanje sa novim tehnologijama je potrebno i poželjno. Međutim, ako se nove tehnologije bez opravdanih razloga integriraju u proizvode dobijamo otpad.

Primjer:

Developer procjeni da mu “Python runtime” može biti od velike pomoći u budućnosti, te stoga daje uputu serviserima da instaliraju “Python” na svaku radnu stanicu kod korisnika.

Iako je prednost ove aktivnosti za developera važna, gubici koje nije uzeo u obzir su sljedeći:

- Serviser gubi dodatno vrijeme na instalaciju softvera koji korisniku nije potreban
- Taj softver nepotrebno zauzima prostor
- Svaki komad softvera je predmet budućeg održavanja i moguća sigurnosna prijetnja

Ovo je tipični primjer otpada u Lean tehnologiji.

10 Zaključak

Agilni manifest je objavljen prije 17 godina. Danas se može reći da su agilne metode postale prije standard nego li novitet u organizaciji softverskih projekata i razvojnih timova. Agilni razvoj prepoznaje specifičnosti razvoja softvera u odnosu na druge inžinjerske grane. Dinamičnost industrije informacionih tehnologija određuje da učenje i usvajanje novih znanja bude ne samo put ka izvrsnosti, nego i uslov opstanka. Agilni razvoj je rezultat traganja za konceptima te činjenice uvažavajući radnu psihologiju. One su stoga koliko tehničke toliko i emocionalne - bave se programerskim tehnikama, ali i motivacijom, ugodnom radnom atmosferom, osjećajem postignuća.

U praktičnom dijelu rada prikazan je razvojni ciklus projekta "F18". Navedeni su motivi, tehnički i poslovni ciljevi, problemi, te postignuti rezultati u razmatranom periodu. "F18" je određen sljedećim faktorima:

- projekat ima 25-godišnju istoriju, što ga u informatici čini starim
- ciljna industrija su finansije, knjigovodstvo i podrška poslovanju
- ciljna klijentela je lokalna, tržište malo (mala kupovna moć)
- razvojni tim je mali (dva čovjeka)

Ovi faktori određuju širi kontekst projekta. Na osnovu njih se mogu procijeniti mogućnosti daljeg razvoja, toka i održivosti projekta. Bilo bi pogrešno ocijeniti razmatrani razvojni period kao period u kome se prešlo (ili nije prešlo) na agilni razvoj softvera. Projekat je u ovom periodu prošao kroz egzistencijalnu krizu. U tom periodu se nije tražio odgovor na optimalan način razvoja, nego odgovor na pitanje "biti ili ne biti". Iako su ovo pitanja na koje bi glavnu riječ trebali voditi ekonomisti a ne informatičari, ovakve situacije i dileme su od temeljne važnosti za projekat i razvojni tim. Ako softverski projekat izgubi elemente održivosti, on propada bez obzira na kvalitet programske kôde i mogućnosti razvojnog tima. Princip 8. autora agilnog Manifesta to naglašava: "*Agilni procesi promoviraju održivi razvoj softvera. Investitori, developeri, korisnici trebaju moći nastaviti učešće u održavanju softvera neograničeno*".

Mnogobrojni nedostaci i problemi projekta "F18" nisu izostavljeni. Mnogi od njih su i dalje prisutni, i po svoj prilici će ostati ograničavajući faktor razvoja. Svaki softverski proizvod je kombinacija mogućeg a ne željenog. On je dobar i funkcioniše u onoj mjeri u kojoj ga takvim **smatraju klijenti koji ga koriste**. Za razvojni tim, odnosno preduzeće koje proizvodi softver, mjera uspjeha su poslovni rezultat, te novi projekti koji nastaju na bazi postojećih. Uvezši u obzir ove parametre, razvojni period 2014-2018. se može ocijeniti "F18" uspješnim.

Za svaku od prezentovanih aktivnosti na "F18" su napravljeni posebni podzaključci u kontekstu primjene agilnih koncepata. Unutar tih podzaključaka najviše referiranja je bilo na "*Lean software development*" metode. Dugogodišnja istorija i stanje projekta na početku razvojnog ciklusa su odredili prioritete, a to su jezikom "lean" bile eliminacija **otpada** i podizanje **kvaliteta** postojeće programske baze.

Najvažniji nedostatak koji je ostao prisutan i nakon završetka razvojnog ciklusa je nedostatak automatskih testova programske baze. To je onemogućilo primjenu TDD praksi³⁹ i veći kvalitet CI procesa. Taj nedostatak je ublažen opcijama “F18 update” i “F18 bug reports”⁴⁰, ali one ipak ne mogu zamijeniti testove. Kreiranje seta testova na “F18” kodu je razmatrano ali je na kraju ipak odbačeno radi malog kapaciteta razvojnog tima naspram velike programske baze, arhitekture aplikacije i ograničenja alata za razvoj⁴¹.

Pozitivni rezultati predhodnog su stvorili pretpostavke da se otvori novi razvojni ciklus. Ranija iskustva i novopostavljeni ciljevi svakako će biti ulazni parametri novog razvojnog ciklusa. Očekivanja su da će rezultati istraživanja iz ovog rada biti od konkretne pomoći razvojnom timu “bring.out”, ali i drugim razvojnim timovima koji rade na sličnim projektima.

39 referenca paragraf: 2.5

40 referenca paragrafi: 8.4, 8.5

41 “harbour” programski jezik nema ugrađenu podršku za formiranje testova kao što to imaju drugi programski jezici

Literatura

AART: James Shore & Shane Waden, The Art of Agile development, 2008

APRAT: Venkat Subramaniam & Andy Hunt, Practices of an Agile Developer, 2006

ASAM: Jonathan Rasmusson, The Agile Samurai, 2014

PROGIT: Scott Chacon & Ben Straub, Pro Git, 2nd Edition, 2014

LEARNAG: Andrew Stellman & Jennifer Greene, Learning Agile, 2015

SEDAWK: Dale Dougherty and Arnold Robbins, sed & awk, Second Edition, 1997

Korišteni softver:

- Windows 10 Home
 - msys2 <https://www.msys2.org>
- Fedora Workstation 28
 - Gnome terminal 3.28
- LibreOffice calc, writer 6.0
- Gimp 2.8
- Pygments 2.2
- Git 2.17
- cloc
 - <https://github.com/AlDanial/cloc>
- draw.io desktop 9.1
- Atom 1.30
- harbour 3.4.6hernad (0ed48dfa0b) (2018-04-05 12:24)
 - <https://github.com/hernad/harbour-core>