UNIVERZITET DŽEMAL BIJEDIĆ U MOSTARU

FAKULTET INFORMACIJSKIH TEHNOLOGIJA MOSTAR

SEMINAR

Agilni *software development*, *Continuous Integration (CI)*

Student: *Ernad Husremović, DL 2792*

Mentor: *mr. Adil Joldić*

ver: 1.0.1

Mostar, novembar 2012.

SADRŽAJ

Abstract

U ovom radu se na bazi konkretnog primjera[1] prezentuje infrastruktura za stalnu integraciju (eng. Continous Integration, CI)

CI infrastruktura je usko vezana za testnu i SCM[2] infrastrukturu(Husremović, 2012a)

Unutar materijala ćemo obraditi implementaciju "Travis Continous Integration" sistema za projekat "F18 knowhow".

Tokom realizacije se koriste sljedeće komponente:

- Github servis, F18 knowhowERP repozitorij - `https://github.com/knowhow/F18_knowhow`
- Travis CI - `https://travis-ci.org`
- Vagrant test environment - `http://vagrantup.com`

Keywords: open source software, OSS, travis, CI, continuous integration, vagrant, github, git

[1] "HOWTO" stil

[2] Source Code management

1. Uvod

1.1. Zašto ”Continuous integration” ?

Mnogi softverski projekti imaju skriveno kašnjenje između trenutka kada developerski tim kaže "Gotovi smo" i trenutka kada je *software* doista spreman za isporuku klijentu.(J.Shore i S.Warden, 2008, str. 183) Kod složenijih projekata se to kašnjenje može rastegnuti na mjesece. Spajanje pojedinih komponenti, kreiranje ”installer”-a, kreiranje inicijalne verzije baze podatataka, kreiranje korisničkog uputstva, itd traže vrijeme. Vrijeme koje razvojni tim često izostavi iz plana. To uzrokuje da se kompletan tim nađe pod stresom. Takvo stanje redovno generira **dodatne** greške i kašnjenje. Kontinuirana integracija (CI) prevenira ovakve situacije. Ona obezbjeđuje da se nakon svakog novog commit-a testira funkcionalnost nove verzija software-a. Time CI omogućava da se odmah nakon razvojnog ciklusa korisniku može isporučiti nova verzija.

> *Glavni cilj CI-a je mogućnost isporuke (eng. deployment) projekta korisniku u svako vrijeme.*

1.2. Travis CI

”Travis CI” je CI sistem koji je besplatan za korištenje za *opensource* projekte. Jedini preduslov je da se sofverski repozitorij koji ”travis” pokriva nalazi na ”github”-u.

Travis funkcioniše tako što se aktivacijom usluge za određenih github projekat, nakon svakog ”commit”-a keira vagrant box[1]. Po kreiranju vagrant box-a, pokreću se komande i vrši konfiguracija box-a u skladu sa konfiguracijskim fajlom .travis.yml.

Interesatno je uočiti da je kompletan projekat spoznoriran od više sofverskih vendora koji na principu sponzorstva dodjeljuju dio svoje infrastrukture projektu[2]

[1] Više o vagrantu materijalu ”Agilni software development, test & deploy infrastructure”(Husremović, 2012b, str. 3)

[2] Organizacija projekta je u potpunosti u duhu ”opensource”-a.

1.3. F18 knowhowERP

F18 je ”client-server” ERP aplikacija:

- klasični (’rich’) klijent pisan u programskom jeziku ”harbour”
- server je PostgreSQL baza podataka

1.3.1. F18 test-case-ovi

Sa stanovišta CI-a problem F18 kao projekta[3] je nepokrivenost source koda testovima. ”Harbour” nema tako dobar ”test framework” kao što je to slučaj sa drugim programskim jezicima[4]. Međutim, čak i rudimentarni test framework hbtest pokazao se dovoljnim za realizaciju F18 ”test case”-ova.

S obzirom da ”F18” korisnik primarno koristi tastaturu za interakciju sa aplikacijom, na ”hbtest” je dodan set funkcija koje omogućavaju simulaciju rada korisnika putem tastature ”keystrokes.prg”

Uz pomoć ”keystrokes.prg” napravljeni su sljedeći integracijski testovi:

- unos novih šifri u šifarnik
- povrat fakture 99-10-7777
- unos fakture sa dvije stavke i njeno ažriranje

Da bi se stekao konačan utisak, pripremljen je video koji prikazuje pokretanje F18 integracijskih testova na testnoj vagrant/virtualbox sesiji:

Link na youtube video ”F18 integracijski testovi”[5]

1.4. Princip CI-a

CI koncept je do kraja jednostavan. CI sistem pokreće skriptu koja definiše uspješnu integraciju sistema. Ta skripta uobičajeno obuhvata dvije faze:

1. ”build” - build sistema
2. ”test” - izvođenje testova na sistemu kreiranom u predhodnom koraku

Ukoliko je ovaj proces uspješan, ”build & test” skripta vraća izlazni kod 0 (exit code = 0, success). U suprotnom, integracija se smatra neuspješnom.

[3] F18 svoje korijene vuče iz devedesetih godina prošlog vijeka, originalno pisan za Clipper/DOS okruženje
[4] `http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks`
[5] `http://www.youtube.com/watch?v=r1SV7GO0MFY`

Ovaj jednostavni ”succes” ili ”fail” koncept daje razvojnom timu brzu informaciju o tome da li su nove promjene uzrokovale probleme u sistemu (regresiju određenih funkcija) ili se integracija novog kôda realizovala uspješno.

1.5. Kvalitet testova

Jasno je da je kvalitetna ”baterija” testova ključna odrednica svrsishodnog CI procesa. Ukoliko je izvorni kôd slabo ”pokriven” testovima, rezultati CI sistema[6] se moraju uzeti sa rezervom.

1.6. ”Technical Debth”

”Technical debt”[7] je količina **trenutnih** loših rješenja u oblasti dizajna i implementacije projekta. Ovo uključuje brze i ”prljave” korekcije koda (eng. dirty hacks) koje omogućavaju da sistem profunkcioniše, ali ga dugoročno čine ranjivim na greške i teškim za održavanje. ”Technical debt” je najčešće uzrokovan lošim programerskim praksama, neiskustvu i generalno lošem nivou znanja unutar razvojnog tima. On se unutar izvornog kôda manifestuje kroz ogromne, nepregledne funkcije (metode), mnoštvo nepotrebnih globalnih varijabli, slijepog (napuštenog) kôda, puno ”TODO” ili ”nisam siguran kako ovo radi ..” komentara.(J.Shore i S.Warden, 2008, str. 41)

1.7. ”Legacy system” i CI

F18 knowhowERP je klasični ”Legacy system”[8] . U vrijeme kada je pravljena većina aplikativnog kôd-a F18, koncept agilnog software developmenta i njemu pripadajuće prakse su bile nepoznate.

Zato je za većinu funkcija F18 ne postoje automatizirani testovi. Agilni pristup u ovakvom slučaju preporučuje sljedeće:

1. Testove postupno uvoditi u kritične funkcije projekta - one koji se najviše koriste i one čija disfunkcionalnost nanosi najveće probleme korisniku[9]

2. Kada se uoči da je potrebno raditi značajne promjene na dijelovima kôda (refactoring), prije promjena napraviti set odgovarajućih testova

[6] Posebno oni koji kažu da je integracija uspješna

[7] Doslovni prevod na bosanski bi bio ”tehnički dug”. Smatramo da se radi jasnoće bolje držati engleskog termina.

[8] `http://en.wikipedia.org/wiki/Legacy_system`

[9] ”Fiskalne funkcije - štampa na fiskalni uređaj” su dobar primjer kritične funkcije projekta

Ovakav pristup će obezbjediti da se ”technical debth” projekta kontinuirano smanjuje ili barem ne uvećava.

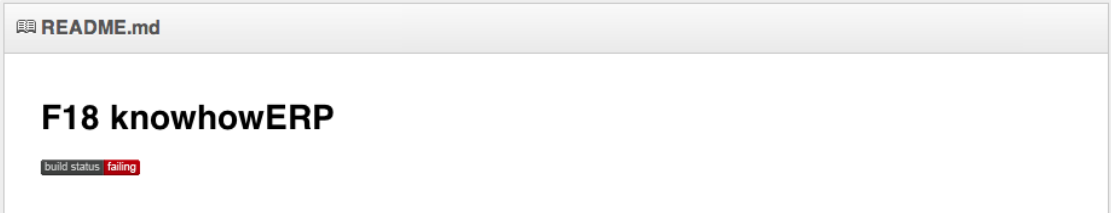
2. Travis

2.1. *Travis* kao indikator ”zdravlja” našeg projekta

Nakon što se ”travis” uspješno podesi, u README.md projekta se dodaje sljedeći kôd:

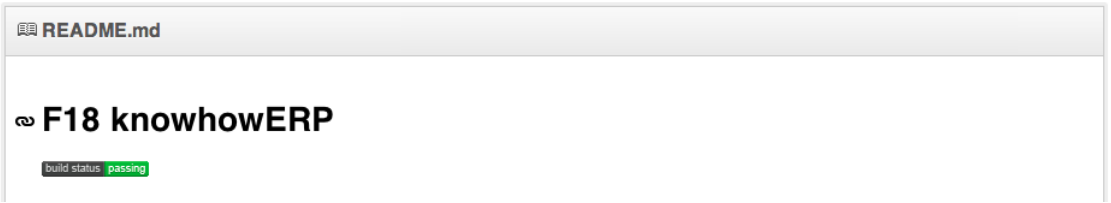
```
[![Build Status](https://secure.travis-ci.org/knowhow/F18_knowhow.png?↩
   branch=master)](https://travis-ci.org/knowhow/F18_knowhow)
```

Na osnovu ovog podešenja, u slučaju neuspješnog ”build”-a imamo ”crveni” - ”failed” status:



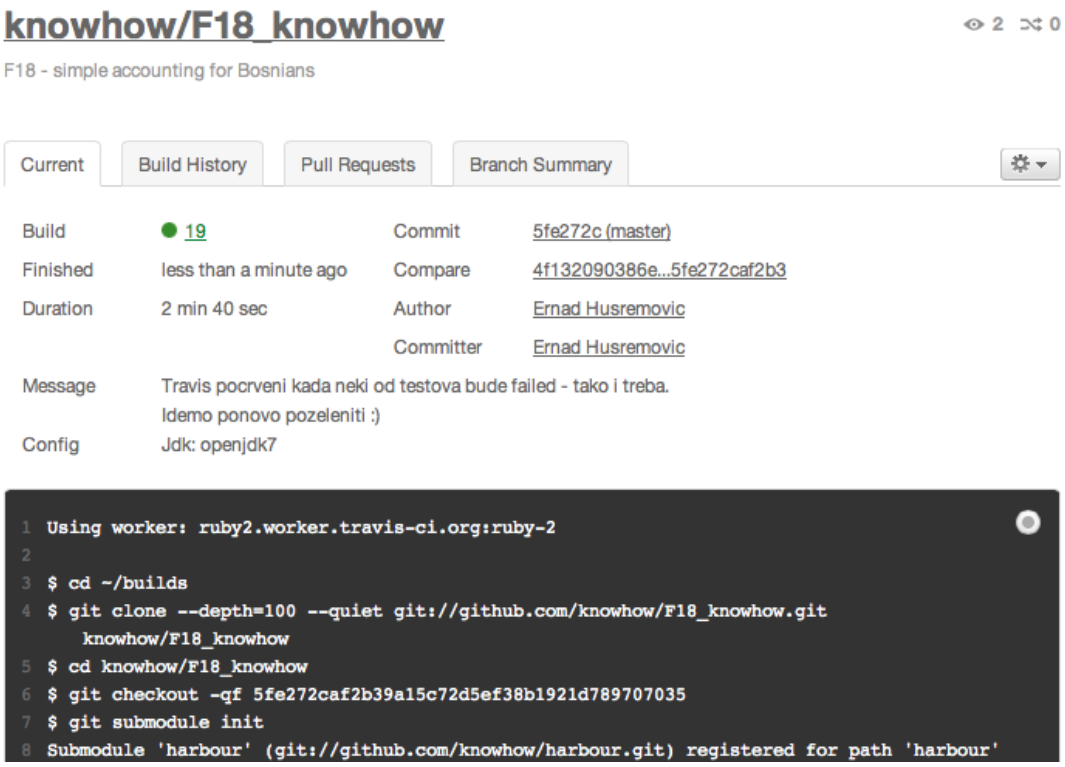
Slika 2.1: travis F18_knowhowERP build neuspješan - ”failing - red build”

A u slučaju uspjepha zelenim statusom:



Slika 2.2: travis F18_knowhowERP uspješno - ”green”

Travis nam daje detaljne informacije o ”build” procesu:

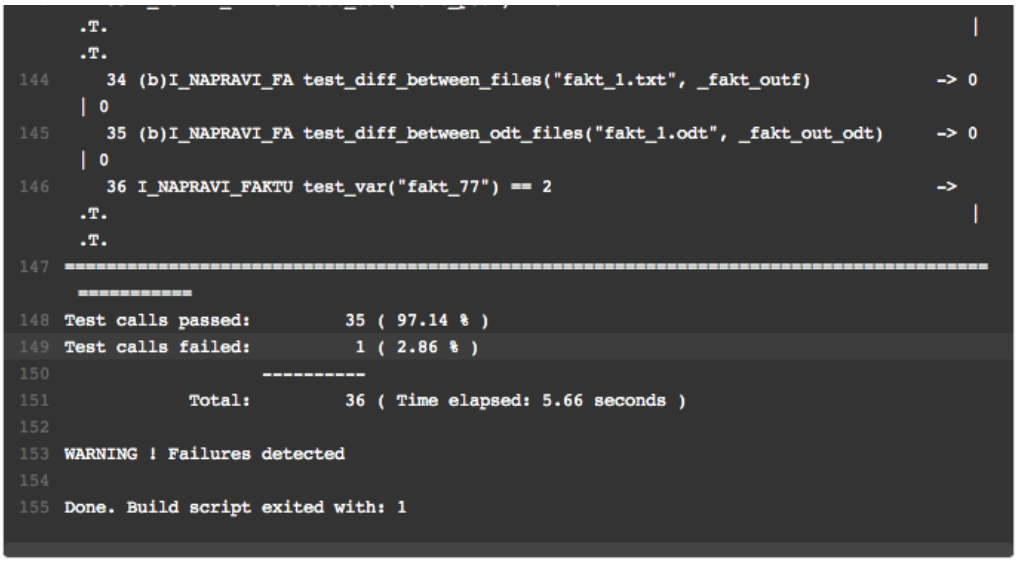


Slika 2.3: travis F18_knowhowERP prvi uspješan build - ”green build”

2.2. *Travis* build log

2.2.1. Neuspješan build

Pogledajmo kako izgleda detaljni izvještaj jednog neuspješog build-a:



Slika 2.4: travis F18_knowhowERP test report - jedan test neuspješan - ”failed build”

Travis daje prikaz svih operacija na CI sistemu tokom ”build & test” procesa unutar ”build log”-a. Primjemio da se kraju log-a pojavljuje ”exit code 1”. Ono odrežuje da da se ”build” označi kao neuspješan.

2.2.2. Uspješnan build

Uspješni build daje ”exit code 0”:

```
    .T.
144   33 I_POVRAT_FAKTUR test_var("fakt_pov") == 0                          ->
    .T.                                                                      |
    .T.
145   34 (b)I_NAPRAVI_FA test_diff_between_files("fakt_1.txt", _fakt_outf)  -> 0
    | 0
146   35 (b)I_NAPRAVI_FA test_diff_between_odt_files("fakt_1.odt", _fakt_out_odt) -> 0
    | 0
147   36 I_NAPRAVI_FAKTU test_var("fakt_77") == 2                           ->
    .T.                                                                      |
    .T.
148 ================================================================================
    ==========
149 Test calls passed:         36 ( 100.00 % )
150 Test calls failed:          0 ( 0.00 % )
151                     ----------
152             Total:         36 ( Time elapsed: 4.57 seconds )
153
154
155
156 Done. Build script exited with: 0
```

Slika 2.5: travis F18_knowhowERP test report prvog uspješnog build-a

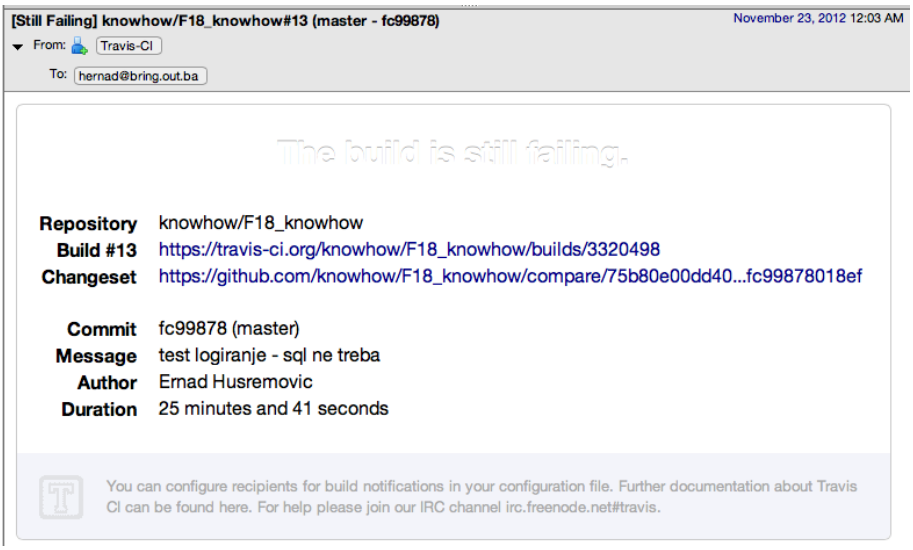
Od 36 testova koji su pokrenuti, svi su uspješno završili

2.3. Email notifikacije

Travis, očekivano, ima sistem email notifikacija. Te notifikacije informišu developere o bitnim događajima u ”build” procesu.

Ono što je za ”travis” email notifikaciju karakterističnо jeste krajnje inteligentan sistem slanja poruka.

Naime, u slučaju uspješnog builda (pod uslovom da je predhodni build bio također uspješan) developer ne prima nikakve notifikacije. Tek u slučaju da build ”pukne”, developer počinje primati informacije ”Build failing”, nakon toga ”Still failing”. Kada build ponovo ”pozeleni”, travis prijavljuje "Fixed !". Ukratko, travis se brine o tome da developera ne ”spamira” nepotrebnim porukama.

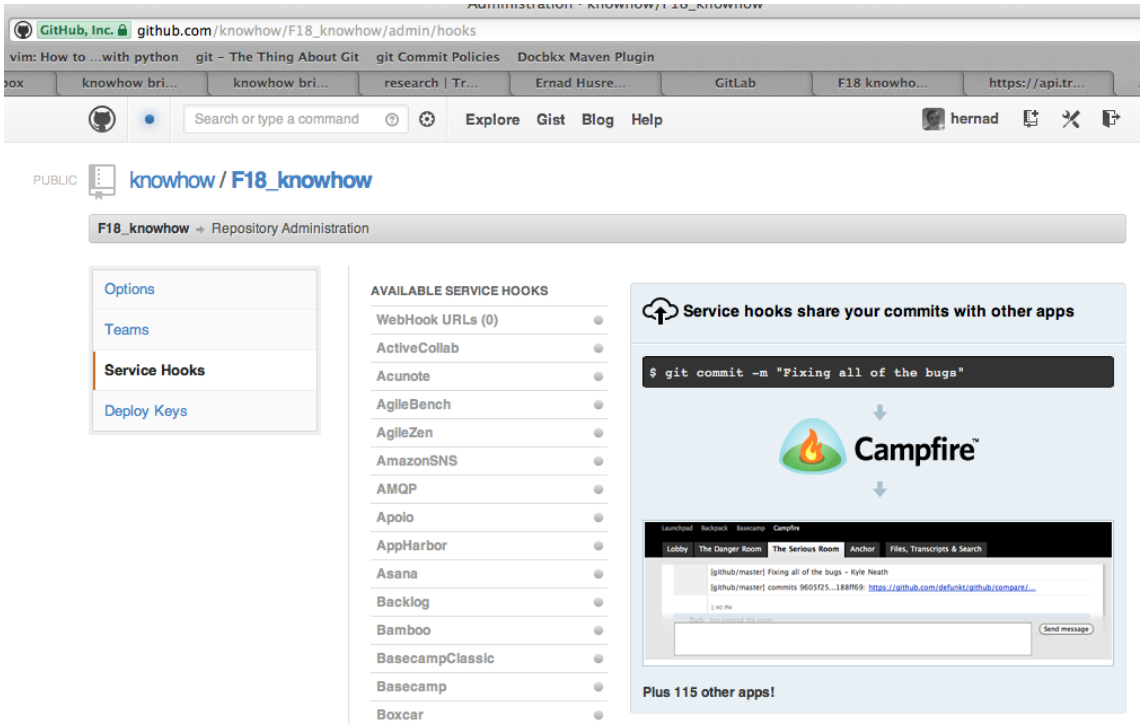


Slika 2.6: Travis email notifikacija

3. Integracija servisa

3.1. *Github* i *travis*

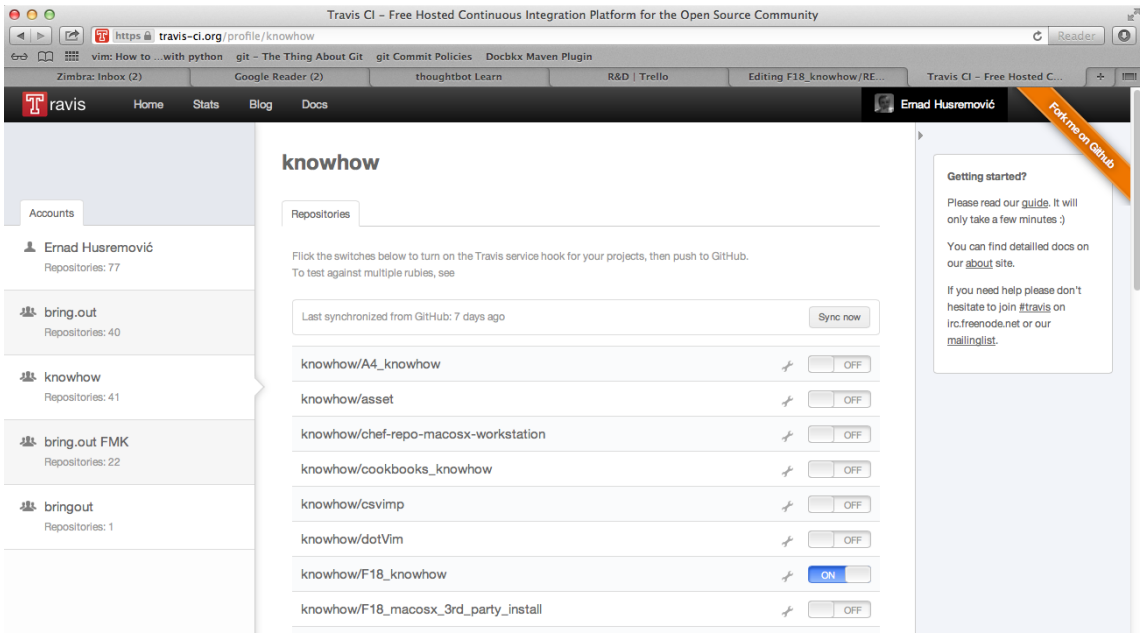
Da bi se ”travis” uopšte koristio, neophodno je da projekat bude hostiran na ”github”-u. Integracija se vrši sistemom Webhook-ova.[1]



Slika 3.1: github web interfejs za podešenja ”Service hook”-ova

Github”service hook” ze podešava unutar travis web interfejsa:

[1] http://en.wikipedia.org/wiki/Webhook. Github umjesto ”Webhooks” koristi termin ”Service Hooks”



Slika 3.2: Travis web interfejs: setup github repozitorija

3.2. *Github - trello* integracija

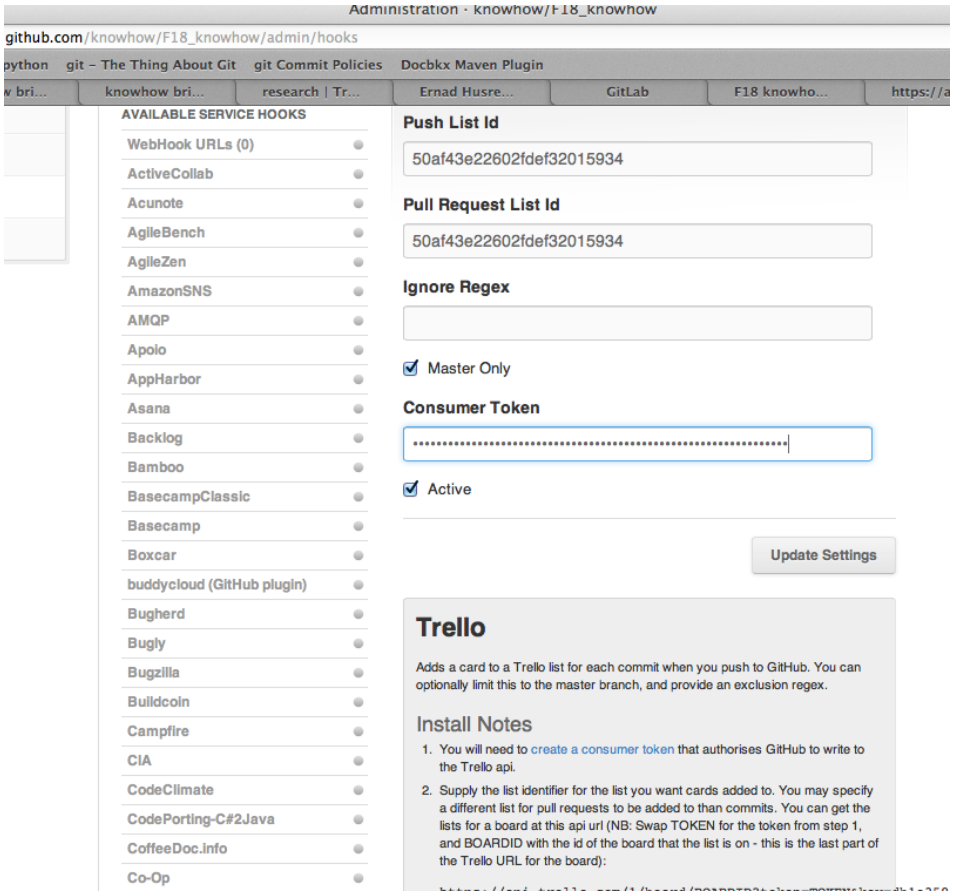
Iako "trello"[2], kao agilni alat za planiranje, strogo gledano ne pripada tematici "CI"-a, prikazaćemo kako se ova integracijom ovog servis realizira efektivno informisanje razvojnog tima.

Slika 3.3: Github: "Service hooks": trello

Podešavamo integraciju F18 trello table (eng. board), za listu "commits"[3]

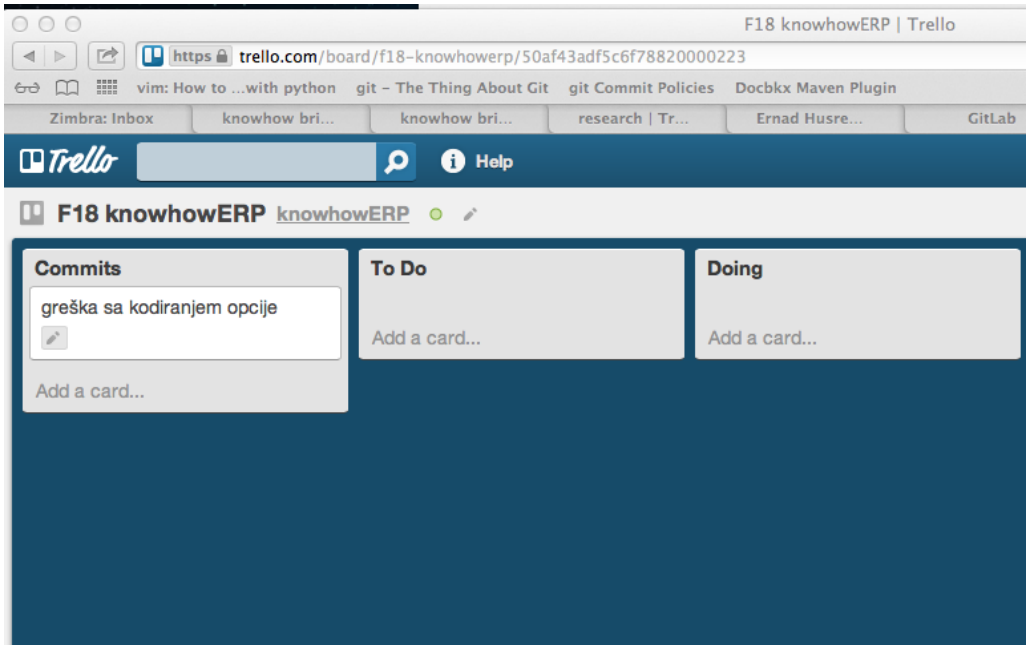
[2] `https://trello.com`

[3] detaljnije `https://github.com/hernad/agile_dev_env/blob/master/CI.md#f18-knowhwhowerp-trello-integracija`



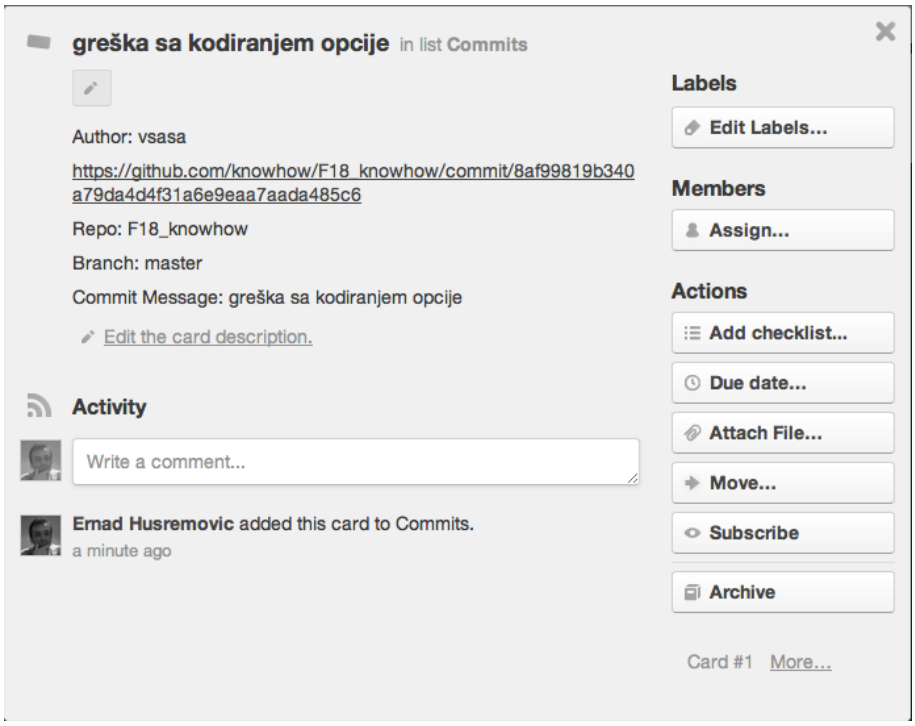
Slika 3.4: Github: Parametri trello integracije

Rezultat integracije je sljedeći:



Slika 3.5: Prikaz ”commit”-a na trello listi ”Commmits”

Nakon svakog ”commit”-a na ”F18_knowhow” repozitorij, kreira se u listi ”Commits” jedna trello kartica (eng. card):



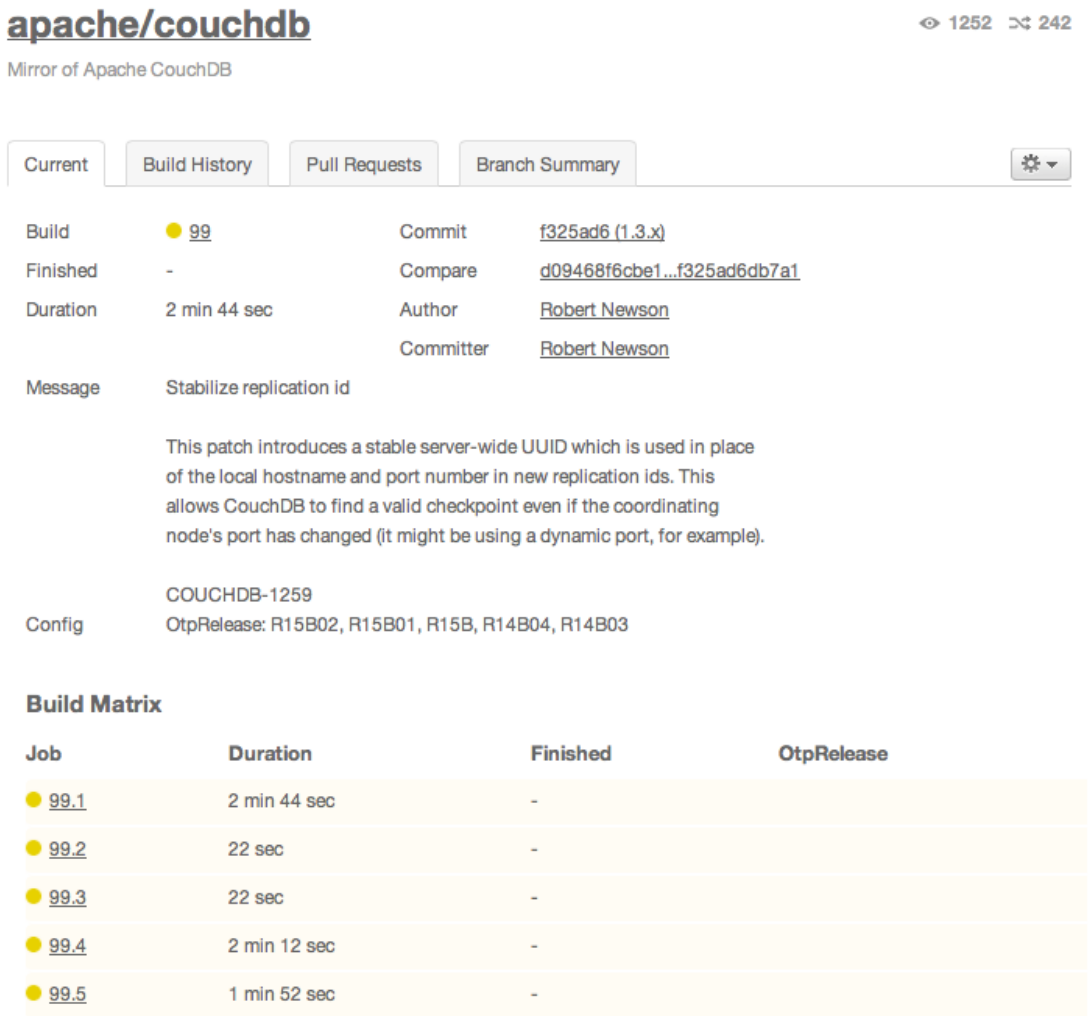
Slika 3.6: Git ”commit” kao ”trello card”

4. Ostalo

4.1. Travis *Build matrix*

U .travis.yml se može definisati više verzija radnog (eng. run-time) okruženja.

Uzmimo za primjer "couchdb" projekat. Kod njega postoji potreba testiranja za više verzija ”erlang” radnog okruženja. Time se formira matrica buildova za svaku verziju erlanga:



Slika 4.1: Travis build matrica

4.2. *Travis* je *linux-only* CI

Ukoliko je projekat ”Windows OS” orjentisan, travis se ne može koristiti kao CI server, iz jednostavnog razloga što su build sesije isključivo ”ubuntu linux”.

4.3. Jenkins CI

U tom slučaju, jenknins-ci je odlično OSS rješenje `http://jenkins-ci.org`. U varijanti ”jenkins”-a je potrebno je obezbjediti sopstvenu ”build” infrastrukturu.

Jenkins je jedino (potpuno) rješenje za multiplatformske projekte kao što je slučaj i sa našim primjer projektom materijala - ”F18_knowhow”.

Jenkins je organizovan java web servlet aplikacija koja se povezuje sa različitim agentima. Tako bi u slučaju ”F18_knowhow” trebali bi uspostaviti sljedeću infrastrukturu

1. jenkins CI server, ubuntu linux server
2. jenkins linux agent, ubuntu linux [1]
3. jenkins windows agent
4. jenkins Mac OS X agent

Agenti obavljaju build operacije po ”instrukcijama” servera. Korisnik na serveru putem web interfejsa, Webhook-ova prati build proces. Primjetimo da je ”client-agents” princip jenkinsa sličan ”build matrix” konceptu ”travis”-a ali omogućava testiranje na različitim operativnim sistemima.

[1] funkciju agenta može obavljati i server

5. Zaključak

Agilni software development snažno promovira praksu *CI*-a. CI značajno utiče na smanjenje *technical debth*-a. Dosljednja primjena CI-a omogućava razvojnom timu da se korisnicima isporučuju nove verzije u najkraćemo roku po okončanju razvojnog ciklusa razvoja software-a. Kvalitetna pokrivenost testovima, koje CI sistem svakodnevno izvršava **prije isporuke** software-a krajnjem korisniku, obezbjeđuje smanjenje isporuka verzija software-a koje unose regresije u softverski sistem.[1]

Na kraju, bitno je uočiti **prožimanje** agilnog razvojnog *toolset*-a:

- *Github*/git SCM alat je vezivna tačka većine operacija i praksi agilnog developera
- *Vagrant* koji smo koristili za uspostavljanje testne infrastrukture(Husremović, 2012b) ali i debugiranje *travis* build-a
- *Continous integration* bez praksi koritšenja SCM-a i testiranja gubi smisao.

[1] Verzije koje se najbolje opisuju kolokvijalnim riječnikom sa: "Majstor jedno napravio, al' dva pokvario"

6. Literatura

Ernad Husremović. *Agilni software development, Git SCM*, 2012a.

Ernad Husremović. *Agilni software development, test & deploy infrastructure*, 2012b.

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

Dodatak A
Priprema F18_knowhow projekta za *travis build*

Podešenje *travisa* je tražilo značajne operacije, iako su željeni F18 testovi bilo spremni.

Build proces obavlja build_travis.sh skripta. Ona obavlja sljedeće:

1. instalira harbour sa *google code download* sekcije knowhow ERP F18 projekta
2. kreira bazu i globalne uloge koje test case-ovi koriste
3. instalira java jod-reports sa google code (eksterni ”dependency” F18 za generaciju ”ODT” dokumenata[1])
4. build F18_test
5. pokreće F18 testove

A.1. *Headless* travis podešenja

Najviše vremena je bilo potrebno da se realizira i testra posljednji korak build procesa pokretanje F18 testova. Razlog za to je činjenica da je F18 GUI aplikacija, a travis build sesija na sebi nema ”X Windows” GUI sistem.

Travis ipak nudi rješenje za ovo putem headless konfiguracije build-a[2]

A.2. Debugiranje *travis* build-a sa *vagrant*-om

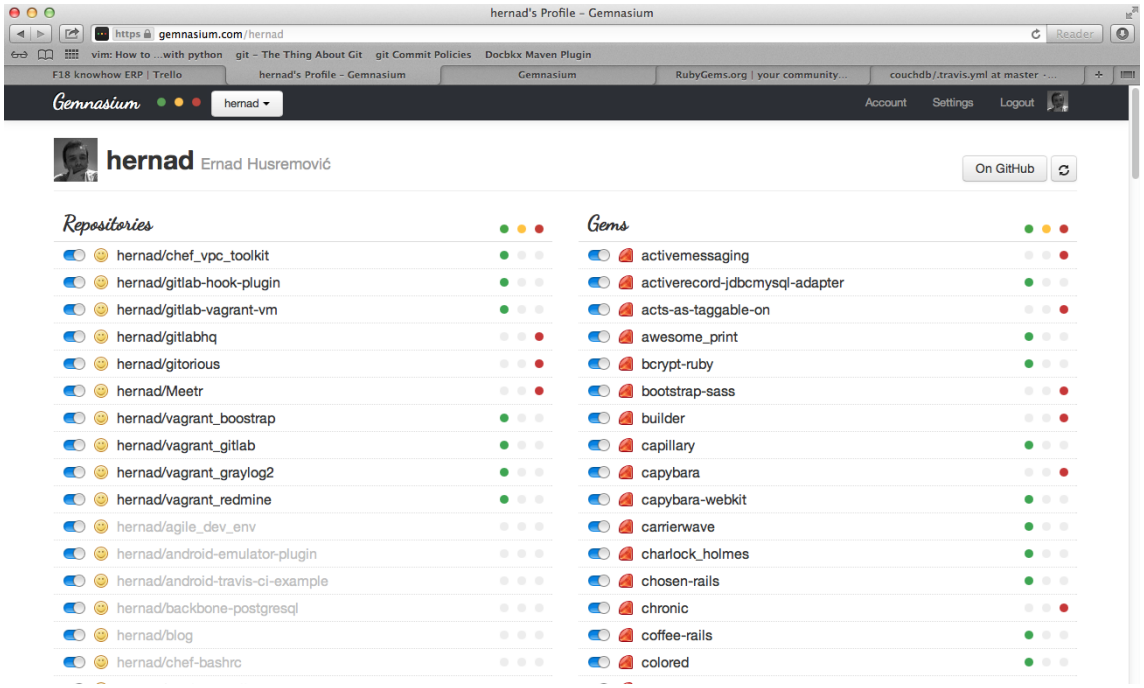
Međutim, i pored svih uputa, bez mogućnosti direktnog testa, u mnogim situacijama nije moguće podesiti uspješan build & test proces. Tu nam pomaže mogućnost da se travis proces pokrene lokalno u ”vagrant” okruženju[3].

[1] oasis ”Open document text” format koje podržavaju ”Openoffice” i ”Libreoffice” aplikacije
[2] `http://about.travis-ci.org/docs/user/gui-and-headless-browsers`
[3] http://ruby-journal.com/debug-your-failed-test-in-travis-ci

Dodatak B
Ostale bilješke

B.1. *Gemnasium* ruby *dependancy management*

Gemnasium[1] je samostalan web servis koji se integriše sa github-om. Orjentisan je isključivo na praćenje zavisnosti između ruby gem-ova[2] koji se koriste u sopstevnim projektima. Iako je orjentisam na "ruby" projekte, njegov koncept je svakako vrijedan pažnje.



Slika B.1: Gemnasium za github "hernad" korisnički račun

B.2. Drugi o *travis*-u

– "Thoughtbot" o travisu

[1] `https://gemnasium.com/questions`

[2] Ruby gems `http://rubygems.org` je standardizovni sistem za pristup dodatnim ruby bibliotekama i aplikacijama