Testiranje programske opreme

Jakob Maležič

E-pošta: jm6421@student.uni-lj.si

Povzetek. V tej seminarski nalogi so predstavljeni različni procesi testiranja programske opreme. Predstavljene so posledice slabega testiranja, tipi testiranja, ki se dandanes najbolj pogosto uporabljajo, glavni poudarek pa je na razlogih zakaj je testiranje sploh potrebno. Podanih je tudi nekaj slabih praks oz. česa si pri testiranju ne želimo in na kaj moramo biti pozorni.

Ključne besede: testiranje, testi enote, integracijski testi, verifikacijski & validacijski testi, code coverage

Software testing

This seminar paper talks about different ways of software testing. It's main focus are reasons for software testing and different types of software testing. It also presents wrong ways of software testing or so called »anti-patterns«.

Keywords: software testing, unit tests, integration tests, verification & validation tests, code coverage

1 UVOD

Testiranje programske opreme je namenjeno ocenjevanju kvalitete programske opreme. S testiranjem lahko objektivno ugotovimo kako dobra je naša programska oprema ter kje so njene slabosti. Za večino aplikacij, nam je samoumevno da delujejo pravilno, na primer aplikacije, ki skrbijo za delovanje jedrskih elektrarn, finančnih inštitucij, borz, kirurških robotov. Če pa bi v takih aplikacijah prišlo do kakršnekoli napake, bi bile lahko posledice katastrofalne. Zato je zelo pomembno pravilno testiranje, ki nas na take napake vnaprej opozori.

2 POSLEDICE SLABEGA TESTIRANJA

Kot že rečeno je testiranje izjemno pomembno, ker lahko programske napake vodijo tudi v velike finančne izgube. Nekaj primerov:

- Aprila 2015 je Bloombergov terminal imel napako, zaradi katere je nastalo 3 bilijone izgube
- Avtomobilsko podjetje Nissan je moralo preklicati več kot milijon avtov zaradi programske napake v senzorjih za zračne blazine (angl. airbag)
- Podjetje Starbucks je bilo primorano zapreti 60% vseh svojih poslovalnic v ZDA ter Kanadi, ker so imeli programsko napako v POS terminalih.
- Zaradi programske napake so nekateri izdelki na Amazonu stali zgolj 1 penny
- Kitajsko letalo je 26. aprila 1994 strmoglavilo zaradi programske napake

3 TIPI TESTIRANJA

3.1 Testi enot (angl. unit tests)

Testi enot so po navadi avtomatični testi, ki jih poganjajo razvijalci programske opreme, da zagotovijo da posamezna »enota« programa pravilno deluje. V funkcijskem programiranju je to po navadi zgolj ena funkcija. V objektno orientiranem programiranju pa je enota velikokrat kar cel razred, lahko pa je tudi posamezna metoda.

3.1.1 Prednosti

Cilj testiranja enot je izolirati vsak del programa ter pokazati da so vsi individualni deli pravilni. Testi enote določajo striktne pogoje, katerim mora del kode zadoščati. Zato imajo nekaj prednosti:

- Testi enote najdejo napake že zgodaj v razvojnem ciklu. Tako programske napake kot manjkajoče dele, ki so specificirani v enoti.
- Proces pisanja podrobnih testov prisili programerja da dobro premisli o vhodih, izhodih in možnih napakah ter s tem bolj točno definira obnašanje enote.
- Cena iskanja napake pred pisanjem kode je veliko manjša kot iskanje enake napake kasneje.
- Kodo je nemogoče ali pa zelo težko testirati če je slabo spisana, zato lahko testiranje prisili programerje da pišejo boljšo kodo.

3.1.2 Test driven development (TDD)

V razvoju, ki temelji na testih, angl. test driven development, ki se pogosto uporablja v »extreme programming« ter »scrum«, so testi enote napisani pred kodo. Tako koda ni sprejemljiva, dokler vsi testi ne delujejo.

Ko se razvija večji del kode, se isti testi poganjajo večkrat. Vsakič ko se koda spremeni ali pa pri vsaki gradnji angl. buildu kode. Če testi ne delujejo, pomeni da imamo napako ali v spremenjeni kodi ali pa v testu. Test nam tako olajša iskanje te napake. Ker taki testi zgodaj v razvoju opozorijo na napake, le te manjkrat pridejo do končnih uporabnikov.

3.2 Integracijski testi (angl. integration tests)

Integracijski testi testirajo integracijo posameznih modulov med sabo. Z njimi preverimo skladnost sistema ali komponente s specificiranimi funkcijskimi zahtevami. Izvajajo se po testiranju enot ter pred validacijskimi testi. Poznamo različne pristope integracijski testov.

3.2.1 Big Bang

Večino razvitih modulov združimo skupaj, da tvorijo končan programski sistem ali pa vsaj večinski del sistema in jih nato uporabimo za integracijske teste. Ta metoda je zelo časovno učinkovita, vendar se lahko hitro zgodi, da testni primeri in njihovi rezultati niso primerno določeni. Tako se lahko integracija hitro zakomplicira.

3.2.2 Bottom Up

»Bottom Up« ali od spodaj navzgor, je metoda integracijskega testiranja, kjer najprej testiramo najenostavnejše komponente, ki jih potem uporabimo za testiranje bolj kompleksnih. Proces ponavljamo, dokler ne testiramo najkompleksnejših komponent. Ta metoda je dobra samo takrat, ko so vse ali pa skoraj vse komponente nekega nivoja pripravljene.

3.2.3 Top Down

»Top Down« ali od zgoraj navzdol, je metoda ki je ravno nasprotna od »Bottom Up«. Najprej so testirane najvišje, oz. najbolj kompleksne komponente individualno, nato pa testiramo manj kompleksne dokler ne pridemo do najbolj osnovnih komponent. Prednost je predvsem ta, da najprej testiramo kritične ter najpomembnejše komponente.

3.3 Verifikacijski ter Validacijski testi

V&V testi preverjajo če programska oprema zadostuje specifikacijam in izpolnjuje svoj namen. Verifikacija pomeni da preverjamo če je naš produkt pravilno izdelan, medtem ko validacija preverja če smo naredili ustrezen produkt.

4 CODE COVERAGE

»Code coverage« oz. »test coverage« je pojem, ki opisuje kolikšen delež programa se preveri s testi, ki so zanj specificirani. Večji kot je code coverage, večji delež programa se je preveril s testi, iz česar lahko pričakujemo manj napak, vendar ne nujno. Lahko imamo 100% code coverage, pa bo program še vedno imel napako, saj je odvisno tudi od tega, kaj testi preverjajo.

5 SLABE PRAKSE

5.1 Testi enot brez integracijskih testov

Nekatere napake lahko zaznajo zgolj integracijski testi. Najbolj očitne so tu napake v povezavi z bazo. Transakcije in druge operacije v zvezi z bazo lahko preverimo samo z integracijskimi testi. Razen če delamo nekaj zelo izoliranega nujno potrebujemo tudi integracijske teste, da zaznamo napake, ki jih testi enot ne zaznajo.

5.2 Integracijski testi brez testov enot

V teoriji bi lahko imeli zgolj integracijske teste vendar v praksi pa so testi enot:

- Lažji za vzdrževanje
- Lažje zaznajo robne primere
- So veliko hitrejši
- Slabe teste enot je veliko lažje popraviti kot integracijske

5.3 Testiranje napačnih funkcionalnosti

Načeloma si želimo da bi naša programska oprema imela 100% code coverage. Vendar v praksi se izkaže da je to zelo težko doseči, hkrati pa to ne pomeni da je naša programska oprema brez napak. V večjih projektih se večkrat izkaže da je lahko že 20% code coverage-a dovolj, vendar teh 20% pokrije 100% kritičnih delov. Kritični deli pa so tisti, ki se velikokrat pokvarijo, posodabljajo ali pa imajo velik vpliv na končnega uporabnika.

6 POVZETEK

Če povzamemo, testi so izredno pomemben del razvoja programske opreme. Brez njih lahko pride do katastrofalnih posledic. Pomembno je da svojo programsko opremo pravilno in dobro testiramo.
Poznamo različne tipe testiranja programske opreme: testi enot, integracijski testi, verifikacijski ter validacijski testi in še mnogo drugih. Vsak tip testov ima svoje prednosti in slabosti, zato je pomembno da skrbimo za dobro ravnotežje med njimi. Prav tako je pomembno da testiramo pravilne dele kode ter se ne oziramo preveč na code coverage, saj nam le ta ne nujno zagotavlja pravilno delovanje.

7 VIRI

[1] MOHARIČ, Tadej, 2015, Testiranje programske opreme [na spletu]. Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko. [Dostopano 21 november 2019]. Pridobljeno s: https://dk.um.si/IzpisGradiva.php?lang=slv&id=55287

[2] Uporabna informatika Vol 26 No 2, , online, https://uporabna-informatika.si/ [Dostopano 21 november 2019].

[3] https://en.wikipedia.org/wiki/Software_testing [Dostopano 21 november 2019].

[4] http://blog.codepipes.com/testing/software-testing-antipatterns.html [Dostopano 21 november 2019].