Prijedlog rješenja za automatizovano generisanje rezultata testiranja Java aplikacija baziranog na integraciji JUnit alata i JavaDoc generatora dokumentacije

Nikola Obradović
Elektrotehnički fakultet Banja Luka
Banja Luka, BiH
nikola.obradovic@etfbl.net

Aleksandar Keleč
Elektrotehnički fakultet Banja Luka
Banja Luka, BiH
aleksandar.kelec@etfbl.net

Sadržaj - **Testiranje predstavlja veoma važan korak u životnom ciklusu softvera i jedan sveobuhvatan proces kome treba posvetiti značajnu pažnju. Pouzdanim softverskim rješenjem može se smatrati samo ono rješenje koje je u toku i nakon razvoja podvrgnuto različitim i veoma iscrpnim mehanizmima testiranja. U ovom radu opisan je alat koji je kreiran sa ciljem da automatizuje proces testiranja Java aplikacija i omogući detaljan prikaz rezultata testiranja ali i dijelova koda koji su učestvovali u pojedinim testovima. Alat je baziran na integraciji JUnit alata za testiranje i JavaDoc generatora dokumentacije Java aplikacija.**

Ključne riječi - Testiranje; JUnit; JavaDoc;

I. UVOD

U životnom ciklusu svakog softverskog proizvoda, testiranje predstavlja veoma važnu ulogu i nezaobilazan korak u procesu razvoja i održavanja. Testiranje softvera je proces analize elementa softvera kako bi se utvrdila razlika između postojećeg stanja softvera i zahtjeva naručioca i kako bi se ustanovile karakteristike softvera. Drugim riječima, testiranje predstavlja pokušaj da se pronađu greške u softveru (eng. *bug*) i detektuje svako neočekivano ponašanje softvera. Veoma je važno da se greške u softveru otkriju u toku razvoja softvera i prije njegove isporuke naručiocu. Svako naknadno otkrivanje grešaka može da prouzrokuje značajne posljedice i velike novčane troškove, naročito kada se radi o softverskim rješenjima koja se koriste u primjenama gdje su pouzdanost i otpornost na otkaze ključni zahtjevi.

Danas postoji veliki broj alata za testiranje koji su bazirani na različitim tehnikama testiranja i prezentovanja rezultata testiranja. U ovom radu pažnja je posvećena testiranju Java aplikacija i prezentovanju rezultata testiranja kao i entiteta koji su testirani. Kada je u pitanju testiranje softvera pisanog u programskom jeziku Java, dominatno okruženje koje se koristi u ove svrhe je JUnit testno okruženje (eng. *framework*). U radu je opisan JUnit alat, opisane su njegove specifičnosti i analizirani su mehanizmi koje koristi prilikom testiranja aplikacija. Opisane su prednosti ovog alata ali i njegovi nedostaci, naročito u pogledu predstavljanja rezultata testiranja i testiranih dijelova koda.

Da bi rezultati testiranja bili što razumljiviji i čitljiviji za testni tim, potrebno ih je predstaviti u odgovarajućem obliku koji je pogodan za brzu interpretaciju od strane testnog tima. Danas postoje različiti generatori izvještaja, koji koriste različite formate za prikaz sadržaja, poput tekstualnog dokumenta, PDF dokumenta, XML dokumenta, HTML stranice, itd. Veliku prednost u odnosu na ostale formate donosi prikaz izvještaja u vidu HTML dokumenta jer ovakav dokument omogućava interakciju sa korisnikom i brzu navigaciju pomoću mreže hiperlinkova. To znači da bi testni tim, u slučaju HTML izvještaja, mogao veoma brzo da se pozicionira na željeni rezultat i da vidi detalje jednostavnim klikom na odgovarajući link. Međutim, JUnit nema podršku za direktnu konverziju rezultata testiranja u HTML dokument, što ovaj alat ograničava u tom pogledu [1]. Umjesto toga, moguće je generisati XML dokument, a onda, korištenjem nekog drugog alata, konvertovati XML dokument u HTML dokument. Pored toga, nije moguće prikazati komentare koji se navode uz metode i koji često nose dosta korisnih informacija, odnosno, nije moguće integrisati JavaDoc dokumentaciju u okviru izvještaja o testiranju.

U ovom radu predstavljen je i opisan JTester, kao alat koji omogućava jedinstven prikaz JUnit testova u vidu HTML dokumenta sa svim pripadajućim podacima, tj. rezultatima testiranja, kao i odgovarajućim komentarima koje su članovi testnog tima pisali uz testove i koji se u izvještaj integrišu iz JavaDoc dokumentacije.

II. JUNIT TESTNO OKRUŽENJE

JUnit predstavlja regresiono testno okruženje za testiranje Java aplikacija, bazirano na jediničnim testovima (eng. *unit testing*) [1]. Ovaj alat se dominantno koristi u procesu testiranja Java aplikacija i omogućava detaljnu analizu svih dijelova aplikacije, pojedinačno i u sprezi sa drugim dijelovima. Prema istraživanju koje je sprovedeno 2013. godine, na uzorku od 10,000 Java projekata na *GitHub*-u, trećina je koristila JUnit biblioteku, što ovu biblioteku čini jednom od najkorištenijih eksternih biblioteka kada su u pitanju Java projekti [2].

Kada je testiranje softvera u pitanju, posebno važan aspekt predstavlja prikaz i interpretacija rezultata testiranja. Iako se pokazao kao dobro rješenje za sprovođenje različitih vrsta testova, JUnit alat ne može da se svrsta u alate koji detaljno i koncizno prikazuju postignute rezultate. U prilog tome ide i veoma štura dokumentacija koja opisuje generisanje izvještaja pomoću ovog alata [3].

JUnit rezultate testiranja predstavlja u vidu generisanog XML dokumenta. Da bi se ovako generisan dokument mogao prikazati i analizirati, potrebno je koristiti poseban alat ili okruženje, poput *Jenkins*-a [4]. Međutim, zbog svoje strukture i načina na koji kombinuje podatke sa gradivnim jedinicama, XML dokument nije pogodan za brzu i efikasnu interpretaciju od strane članova testnog tima, pa je jasno da je ovakav dokument potrebno konvertovati u neki drugi format. JUnit nema podršku za direktno generisanje izvještaja u vidu HTML dokumenta, pa je za ovu namjenu potrebno koristiti odgovarajuću eksternu biblioteku, kao što je *Ant* [5]. Da bi se *Ant* iskoristio za generisanje HTML izvještaja, u JUnit projekat potrebno je uključiti odgovarajuću *Ant* skriptu. Primjer ovakve skripte dat je u poglavlju IV.

III. JUNIT I JAVADOC

Pisanje dokumentacije izvornog koda predstavlja još jedan važan korak u izradi kvalitetnog softverskog rješenja, koje olakšava saradnju članova razvojnog tima ali i tima za testiranje. Dobro dokumentovan softver omogućava brzo razumijevanje njegovih segmenata, efikasne izmjene i dorade, efikasno testiranje i otkrivanje grešaka u kodu (eng. *debugging*), kao i višestruku upotrebu takvog koda od strane različitih razvojnih timova.

Prilikom predstavljanja rezultata testiranja, poželjno je prikazati i informacije o dijelovima koda koji su učestvovali u određenim testovima, tj. nazive klasa, metoda, objekata, itd. Kada je u pitanju generisanje dokumentacije izvornog koda Java aplikacija, obično se misli na JavaDoc, kao najpoznatiji i najkorišteniji generator dokumentacije. JavaDoc generiše programski API, tj. dokumentaciju izvornog koda, u vidu HTML dokumenta, upravo zbog pogodnosti koje donosi ovaj format dokumenta, a koje su opisane ranije u radu [6].

Upravo zbog ovih karakteristika, poželjno bi bilo integrisati JavaDoc komentare u okviru JUnit testova i u generisanom HTML izvještaju sjediniti podatke koje generišu ovi alati. Uključivanjem dokumentacije u izvještaj o testiranju dobija se šira slika o tome šta se zapravo testira i razloge zbog kojih se testira. Tipično, u JUnit izvještaju navodi se naziv klase i metode nad kojom su pokrenuti testovi, kao i rezultat izvršavanja metode koji može da ima dvije vrijednosti, „uspješno“ i „neuspješno“. Ovi podaci članovima testnog tima ne daju dovoljno detalja i ne daju odgovore na pitanja šta je tačno testirano, koji su očekivani rezultati testiranja, kada se javljaju izuzeci, šta predstavljaju ulazne vrijednosti testa, itd. Osnovni cilj JavaDoc dokumentacije je da omogući čitaocu potpuno razumijevanje određenog entiteta bez ulaženja u detalje implementacije i analiziranja izvornog koda.

Postoji nekoliko alternativnih načina da se u okviru JUnit izvještaja prikažu dodatne informacije, bez korištenja JavaDoc izvještaja. Jedan od njih je korištenje *Maven Surefire Plugin*-a u kombinaciji sa klasom *RunListener* koja omogućava čitanje sadržaja proizvoljno definisane anotacije [7]. Međutim, ovakav pristup ne donosi potrebnu fleksibilnost koju obezbjeđuje JavaDoc.

IV. POSTOJEĆA RJEŠENJA

Iako, prema nekim izvorima, postoji još od 1998. godine [8], JUnit još uvijek nije pokazao pravi potencijal kada je u pitanju prezentovanje rezultata testiranja i njihova integracija sa drugim tipovima dokumenata. Ovdje se prvenstveno misli na nemogućnost jednostavne integracije JUnit rezultata i JavaDoc dokumentacije. Iz tog razloga, danas postoji nekoliko rješenja koja imaju za cilj da omoguće jednostavniju reprezentaciju rezultata testiranja kao i prevazilaženje nekih nedostataka JUnit alata. Neka od njih su bazirana na JUnit-u a u nekima je razvijen potpuno novi mehanizam za testiranje i generisanje izvještaja testiranja. U nastavku su opisana neka od postojećih rješenja.

A. *JUnitReport*

JUnitReport je alat razvijen sa ciljem da omogući konverziju XML dokumenta, koji generiše JUnit, u HTML dokument i tako omogući stilizovan prikaz rezultata testiranja i pojednostavi navigaciju kroz pojedine testove. *JUnitReport* je baziran na upotrebi *Ant* biblioteke za konverziju XML dokumenta u HTML, kao i određenog broja eksternih biblioteka, kao što je *Xalan* [9]. Osim osnovne namjene, ovaj alat ne donosi dodatne mogućnosti i ne omogućava integrisanje JUnit rezultata sa JavaDoc dokumentacijom. Takođe, prilikom korištenja *JUnitReport*-a, korisnici se često susreću sa problemima vezanim za nekompatibilnost verzija Java biblioteke i biblioteka koje koristi ovaj alat [10].

B. *Maven Surfire Report Plugin*

Maven Surfire Report Plugin je još jedan alat čiji je cilj da omogući generisanje HTML dokumenta sa rezultatima JUnit testova. Za izvršavanje svojih zadataka, ovaj alat koristi *maven* biblioteku, što znači da se može koristiti samo u okviru Java projekata koji su bazirani na ovom alatu za prevođenje i generisanje programa (eng. *build tool*) [7]. *Maven Surfire* takođe nema podršku za integraciju JUnit rezultata i JavaDoc dokumentacije. Međutim, za prikaz dodatnih informacija o testiranim entitetima, ovaj alat nudi alternativu u vidu korištenja korisnički definisane anotacije i klase *RunListener*, za kreiranje posebnog izvještaja koji uključuje i sadržaj kreirane anotacije [11]. Iako na ovaj način izvještaj o testiranju može da se upotpuni dodatnim komentarima, ipak nije postignuta fleksibilnost koju donosi JavaDoc i kompatibilnost sa drugim alatima i bibliotekama koje, u najvećem broju slučajeva, imaju podršku za JavaDoc generator.

C. *TestNG*

TestNG je testno razvojno okruženje razvijeno kao alternativa JUnit alatu sa ciljem da se olakša proces testiranja i uvedu dodatne funkcionalnosti koje JUnit ne podržava. Osim

što omogućava veću fleksibilnost i donosi neke dodatne funkcionalnosti, poput parametrizovanih testova, grupnih testova i dodatnih anotacija, *TestNG* nudi i veću fleksibilnost kada je u pitanju generisanje izvještaja sa rezultatima testiranja. Dok JUnit rezultate svih testova nudi u vidu XML dokumenta, *TestNG* nudi mogućnost generisanja izveštaja u različitim formatima, što, između ostalog, podrazumijeva i generisanje HTML dokumenta sa rezultatima [12]. Iako donosi brojna poboljšanja, ni *TestNG* nema podršku za integraciju sa JavaDoc dokumentacijom. Umjesto toga, ovaj alat nudi alternativni pristup u generisanju strukturirane dokumentacije, koja je slična onoj koju generiše JavaDoc [13]. Međutim, ni ovaj pristup ne garantuje kompatibilnost sa drugim alatima, koji su uglavnom orjentisani ka JavaDoc-u. Osim toga, korištenje JUnit alata znači održavanje kompatibilnosti sa preko 60% Java projekata na *GitHub*-u, za razliku od oko 6% projekata koji koriste *TestNG* [12].

D. *JUnit 5*

Krajem 2016. godine objavljena je verzija 5 JUnit alata, koja je donijela značajne novine u odnosu na starije verzije. Između ostalog, novina koja se tiče poboljšanja kada je u pitanju generisanje izvještaja, odnosi se na uvođenje anotacije *DisplayName* koja omogućava obogaćivanje izvještaja o testiranju sadržajem koji se navodi u okviru ove anotacije [14]. Na ovaj način riješen je problem koji je bio evidentan u ranijim verzijama JUnit-a i koji je JTester alat, koji je opisan u narednom poglavlju, riješio za sve verzije JUnit-a. Međutim, za razliku od JTester-a, ni najnovija verzija JUnit-a ne omogućava integraciju rezultata testiranja sa JavaDoc dokumentacijom.

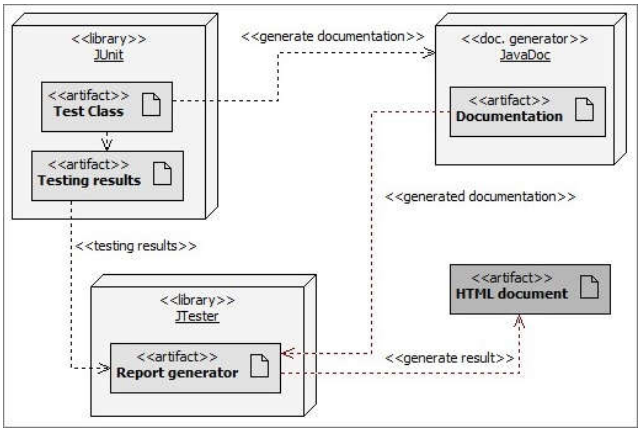
Iako postoji veliki broj eksternih biblioteka, alata i dodataka za razvojna okruženja (eng. *plugin*), razvijenih sa ciljem da se prevaziđu nedostaci JUnit alata kada je u pitanju generisanje izvještaja testiranja, ipak, nijedno od njih ne adresira na pravi način problem integracije dokumentacije sa rezultatima testiranja.

V. JTESTER ALAT

JTester je alat razvijen sa ciljem rješavanja detektovanog problema koji se ispoljava kao nemogućnost integracije JavaDoc dokumentacije u sklopu JUnit izvještaja. U sklopu izvještaja koji JUnit generiše u obliku XML dokumenta neki od standardnih elemenata su: naziv klase u kojoj je testna metoda, naziv metode u kojoj je kôd specifičnog testa, rezultat testa i izuzetak koji se generisao ukoliko test nije bio uspješan. Trenutno podrazumjevano rješenje u sklopu izvještaja ne generiše JavaDoc dokumentaciju, koja bi bila od velike pomoći za dalji razvoj i održavanje testova.

Arhitektura predloženog rješenja data je na Sl. 1 u vidu dijagrama raspoređivanja (eng. *Deployment diagram*).

Prije korištenja datog rješenja potrebno je generisati dokumente koji predstavljaju ulaz za JTester. Prvi dokument koji je potrebno kreirati je HTML dokument u kojem se nalaze rezultati testiranja, a drugi dokument (dokumenti) predstavlja generisanu JavaDoc dokumentaciju za klase u kojima se nalaze testovi.



Slika 1. Arhitektura predloženog rješenja

Za prvi dokument prvo se vrši pokretanje JUnit testova te generisanje XML izvještaja sa rezultatima testova. U našem slučaju navedeno je postignuto korištenjem Eclipse IDE-a (eng. *Integrated Development Environment*).

Generisani XML izvještaj se nakon toga konvertuje u HTML dokument. Za navedeno je korištena *Ant* eksterna biblioteka. Na Sl. 2 data je *Ant* skripta kojom se konvertuje XML izvještaj u HTML dokument. Potrebno je napomenuti i da se tom prilikom generiše više HTML dokumenata. Od svih generisanih dokumenata, dokument koji se koristi za dalju konverziju i u kojem se nalaze rezultati svih testova, dat je pod nazivom *all-tests.html.* Uz dokument je potrebno zadržati i datoteku sa *CSS* (eng. *Cascading Style Sheets*) stilovima, a sve sa ciljem zadržavanja određenog formata i izgleda dokumenta. Za pokretanje skripte takođe je korišten Eclipse IDE.

```
<project name="jTester" default="gen" basedir=".">
<description>
    Generate the HTML report from JUnit XML files
</description>
 <target name="gen">
    <property name="genReportDir"
         location="${basedir}/unitTestReports"/>
    <delete dir="${genReportDir}"/>
    <mkdir dir="${genReportDir}"/>
    <junitreport todir="${basedir}/unitTestReports">
       <fileset dir="${basedir}/reports">
          <include name="**/*.xml"/>
       </fileset>
       <report styledir="reportstyle" format="frames"
                todir="${genReportDir}/html"/>
    </junitreport>
 </target>
</project>
```

Slika 2. *Ant* skripta za konvertovanje XML dokumenta u HTML dokument

Pored kreiranog HTML dokumenta potrebno je generisati i JavaDoc dokumentaciju. Za generisanje JavaDoc dokumentacije takođe je korišten Eclipse IDE. Kao ulazni

dokumenti za JTester izdvajaju se oni dokumenti čiji naziv odgovara nazivu klase u kojoj su metode sa testovima (ovo je poželjno učiniti ali ne i obavezno jer JTester na osnovu naziva klase prepoznaje koje dokumente treba koristiti za dalju analizu).

Kako testiranje podrazumijeva sprovođenje različitih grupa testova, ali i iz razloga što za jedan softver može biti kreirano i na stotine testova, poželjno je da se testne metode grupišu po Java klasama koje predstavljaju jednu grupu testova. Stoga je JTester koncipiran tako da generiše izvještaj za jednu ali i za više testnih klasa.

JTester je kreiran kao *jar* (*eng. Java ARchive*) biblioteka koja se može izvršiti uz pomoć komandne linije na sljedeći način:

```
java -jar JTester.jar  arg1, arg2, arg3
```

Argumenti `arg1, arg2` i `arg3` predstavljaju putanje do JUnit HTML dokumenta, JavaDoc dokumenata i datoteke u koju će biti sačuvan novi dokument, respektivno.

Nakon izvršavanja, alat prvo otvara JUnit HTML dokument te vrši parsiranje istog uz pomoć eksterne biblioteke *jsoup* (Java biblioteka namijenjena za parsiranje HTML dokumenata), nakon čega se parsiran dokument smiješta u radnu memoriju. Prvo se preuzima naziv klase iz prve kolone rezultata testova. Nakon toga JTester pokušava da otvori HTML datoteku sa istim nazivom kao što je naziv navedene klase, a koja bi se trebala nalaziti na putanji za JavaDoc dokumentaciju. Ukoliko HTML datoteka sa takvim imenom postoji, ista se takođe parsira uz pomoć biblioteke *jsoup*. Pri tome se u dokumentu pronalazi JavaDoc dokumentacija za klasu i metode. Pronađena dokumentacija se postavlja u dvije heš mape, pri čemu je jedna heš mapa namjenjena za dokumentovanje klase, a druga za dokumentovanje metode. Ključevi heš mapa su naziv klase i naziv metode, respektivno.

Kako su nazivi klasa i metoda isti u oba dokumenta, u JUnit dokumentu se dodaju (apenduju) komentari za datu klasu i metode klase, a na osnovu odgovarajućih ključeva u mapi. Komentari se dodaju kao dodatni HTML tagovi. Kada je završeno dodavanje komentara za jednu klasu, JTester u originalnom dokumentu traži narednu klasu (ukoliko postoji) te ponavlja prethodno opisani postupak za tu klasu.

Na kraju, JTester vrši kreiranje HTML datoteke i upis u istu novokreiranog HTML sadržaja. Na Sl. 3 dat je primjer dokumentovane testne metode. Na Sl. 4 prikazan je dio dokumenta za spomenutu metodu prije pokretanja JTester alata (izvorni JUnit HTML dokument), a na Sl. 5 prikazan je isti dokument ali sa integrisanom JavaDoc dokumentacijom.

```
/**
 * SELECT COUNT(b1.b1attr1) FROM B1 b1
 */
@Test
public void select_count_var(){
                .
                .
                .
        }
```

Slika 3. Primjer dokumentovane testne metode

select_count_var	Success

Slika 4. Dio dokumenta sa testnom metodom prije pokretanja JTester alata

select_count_var SELECT COUNT(b1.b1attr1) FROM B1 b1	Success

Slika 5. Dio dokumenta sa testnom metodom poslije pokretanja JTester alata

Pregled generisanog izvještaja može se izvršiti u internet pretraživaču. Ukoliko određena metoda u testovima ne posjeduje JavaDoc specifikaciju, za istu neće biti dodani komentari iz JavaDoc-a.

Razvijen sa ciljem da riješi nedostatke postojećih alata koji su opisani u prethodnom poglavlju, JTester se pokazao kao rješenje koje je poslužilo svrsi. Međutim, i ovaj alat ima odgovarajuća ograničenja i nedostatke, o čemu će biti riječi u narednom poglavlju.

VI. Ograničenja i budući rad

JTester, slično kao i mnogi alati koji se koriste za ove namjene, je rješenje bazirano na upotrebi *Ant* biblioteke za konverziju XML dokumenta u HTML. Prema tome, upotrebljivost ovog alata vezana je za mogućnost korištenja *Ant* biblioteke u projektu za koji se pišu testovi. Drugim riječima, samo projekti koji koriste ovu biblioteku, mogu da generišu odgovarajući HTML izvještaj koji predstavlja ulaz za JTester alat. Dakle, predloženo rješenje funkcioniše samo ako mu je prosljeđen HTML dokument koji je generisan uz pomoć *Ant*-a, te u koji se, nakon toga, uz pomoć JTester-a dodaje JavaDoc dokumentacija.

Stoga se, kao prvi naredni korak u poboljšanju JTester alata, nameće ideja da se XML dokument, koji generiše JUnit, direktno parsira, te da JTester sam generiše HTML dokument koji će biti istog formata kao HTML dokument koji se generiše pomoću *Ant* skripte, ali sa integrisanom JavaDoc dokumentacijom.

Pored toga, u planu je kreiranje dodatka za Eclipse IDE, sa kojim bi se čitav proces dodatno automatizovao i ubrzao. Na taj način bi se krajnjem korisniku olakšalo korištenje JTester alata, gdje bi on, umjesto uključivanja i konfigurisanja biblioteke, jednostavno pozivao ugrađenu funkcionalnost razvojnog okruženja. Takođe, postoji želja da se napravi dodatna analiza kojom bi se ustanovilo koje još korisne informacije se mogu integrisati u izvještaj kako bi nova verzija JTester alata ponudila što više novih mogućnosti i dodatno uticala na brzinu i efikasnost procesa testiranja softvera i evaluacije rezultata testiranja. Osim toga, u planu je implementacija i validacije izvornog koda, gdje će korisnik biti upozoren ukoliko na odgovarajućem mjestu nije naveo JavaDoc komentar ili navedeni komentar nije validan, a sve sa ciljem izrade kvalitetnog softvera.

VII. Zaključak

U današnjem svijetu informaciono-komunikacionih tehnologija, programeri i krajnji korisnici softvera troše dosta

dragocjenog vremena prilikom analize semantike pojedinih primjera upotrebe. Dobro dokumentovani izvještaji i kôd garantuju lakše održavanje softvera, ali i automatizaciju drugih procesa. Jedan od primjera slabo dokumentovanih izvještaja je generisani JUnit HTML izvještaj. U takvom izvještaju izuzetno je teško prepoznati o kojem specifičnom testnom slučaju se radi te koja je bila funkcija istog, naročito kada se radi o velikom broju testnih slučajeva.

Iako prepoznato kao jedno od vodećih testnih okruženja Java platforme, JUnit nema adekvatnu podršku za generisanje detaljnih izvještaja sprovedenih testova. Trenutni izvještaji ne zadovoljavaju potrebe programera i klijenata te kao takvi moraju se unaprijediti.

Predloženo rješenje nastalo je kao potreba da se riješe problemi prepoznati u praksi i osnovni cilj ovog alata je bio da riješi ranije opisani nedostak u integraciji dokumentacije sa rezultatima testiranja. Uz pomoć JTester-a lakše se analiziraju postojeći testovi te je lako ustanoviti koje korekcije na testovima je potrebno sprovesti kako bi se ispravilo uočeno neočekivano ponašanje softvera, te koje dodatne testove je potrebno napisati za kompletiranje procesa testiranja.

ZAHVALNICA

Posebno se zahvaljujemo firmi SOL (*Serbian Object Laboratories*) koja nam je stvorila preduslove za pisanje ovog rada.

LITERATURA

[1] http://junit.org/, posljednji pristup: 15.01.2017.

[2] "We Analyzed 30,000 GitHub Projects – Here Are The Top 100 Libraries in Java, JS and Ruby", posljednji pristup: 17.01.2017.

[3] http://junit.org/junit4/project-reports.html,posljednji pristup: 20.01.2017.

[4] http://llg.cubic.org/docs/junit/, posljednji pristup: 18.01.2017.

[5] "How do I use Ant to create HTML test reports?", posljednji pristup: 20.01.2017.

[6] http://agile.csc.ncsu.edu/SEMaterials/tutorials/javadoc/, posljednji pristup: 20.01.2017.

[7] http://maven.apache.org/surefire/maven-surefire-plugin, posljednji pristup: 20.01.2017.

[8] https://shebanator.com/2007/08/21/a-brief-history-of-test-frameworks/, posljednji pristup: 21.01.2017.

[9] https://ant.apache.org/manual/Tasks/junitreport.html, posljednji pristup: 20.01.2017.

[10] https://twasink.net/2004/07/20/ant-162-junitreport-doesnt-work-on-java-141/, posljednji pristup: 22.01.2017.

[11] Maven Surefire - Using custom listeners and reporters, posljednji pristup: 24.01.2017.

[12] http://blog.takipi.com/junit-vs-testng-which-testing-framework-should-you-choose/, posljednji pristup: 24.01.2017.

[13] https://blogs.oracle.com/brewing-tests/entry/use_testng_to_create_test, posljednji pristup: 24.01.2017.

[14] https://junit.ci.cloudbees.com/job/JUnit5/javadoc/, posljednji pristup: 24.01.2017.

ABSTRACT

Testing is a very important step in the software life cycle and a comprehensive process to which should be given considerable attention. Reliable software solution can be considered only the solution that is, during and after the development process, exposed to a different and very detailed testing mechanisms. This paper describes a tool that is designed to automate the process of testing Java applications and provide a detailed overview of the test results as well as pieces of code that took part in individual tests. The tool is based on the integration of JUnit testing framework and JavaDoc documentation generator.

THE PROPOSAL OF SOLUTION FOR AUTOMATED GENERATION OF TESTING RESULTS OF JAVA APPLICATIONS BASED ON THE INTEGRATION OF JUNIT TESTING TOOL AND JAVADOC DOCUMENTATION GENERATOR

Nikola Obradovic, Aleksandar Kelec